**Task 1:**

* **Story Action:** Generate the first paragraph of a new fantasy story titled "The Whispering Woods Chronicle."
  + *LLM Prompt Example: "Write the opening paragraph (around 100 words) for a fantasy story called 'The Whispering Woods Chronicle' about a mysterious map found by a young villager."*
* **Git Actions:**
  + Create a new folder on your computer named whispering-woods-chronicle.
  + Open your terminal/command prompt and navigate into this folder (cd whispering-woods-chronicle).
  + Initialize a new Git repository here: git init
  + Create a new text file named chapter1.txt.
  + Paste the LLM-generated paragraph into chapter1.txt and save it.
  + Check the status of your repository: git status (You should see chapter1.txt as untracked).
  + Add the new file to the staging area: git add chapter1.txt
  + Check the status again: git status (You should see chapter1.txt as a new file to be committed).
  + Commit your first change: git commit -m "Initial draft of Chapter 1 opening"
  + View the commit history: git log
* **What you'll learn:** init, status, add, commit, log.

Task 2:

* **Story Action:** Generate a short character description (name, appearance, one key trait) for the young villager who found the map.
  + *LLM Prompt Example: "Describe a young villager named Elara, who is brave and curious, and has just found a mysterious map. About 50 words."*
* **Git Actions:**
  + Create a new file named characters.txt.
  + Paste Elara's description into characters.txt and save.
  + Check the status: git status
  + Stage the new file: git add characters.txt
  + Commit the change: git commit -m "Add character bio for Elara"
  + View the log: git log --oneline (a shorter log view)
* **What you'll learn:** Adding new files to an existing repository, more add and commit.

Task3 :

* **Git Actions:**
  + Open chapter1.txt.
  + Append the new paragraphs to the existing content and save.
  + Check the status: git status (You should see chapter1.txt as modified).
  + See the exact changes you made: git diff (Press 'q' to exit the diff view).
  + Stage the modified file: git add chapter1.txt
  + Commit the changes: git commit -m "Expand Chapter 1 with Elara's decision"
* **What you'll learn:** Modifying existing files, diff.

* **Story Action:** Generate two more paragraphs to continue Chapter 1, where Elara decides to follow the map.
  + *LLM Prompt Example: "Continue the story from 'The Whispering Woods Chronicle' where Elara has found a map. Write two more paragraphs where she examines the map and decides to venture into the woods."*

Task4:

* **Story Action:** What if Elara had a companion? Let's explore this idea without changing our main story yet. Generate a description for a loyal animal companion (e.g., a wolf, an owl).
  + *LLM Prompt Example: "Describe a loyal wolf companion named Shadow for Elara in 'The Whispering Woods Chronicle'. About 50 words."*
* **Git Actions:**
  + Create a new branch for this idea: git branch add-companion
  + Switch to the new branch: git checkout add-companion (or git switch add-companion)
  + Confirm you're on the new branch: git branch (your current branch will have a \*)
  + Open characters.txt and add the companion's description. Save.
  + Stage and commit the change on this branch:
    - git add characters.txt
    - git commit -m "Introduce Shadow, the wolf companion"
  + Switch back to your main branch: git checkout main (or git switch main)
  + Open characters.txt. Notice Shadow is NOT there. This is because that change is only on the add-companion branch.
* **What you'll learn:** branch, checkout/switch. Understanding how branches isolate work.

Task5:

* **Story Action:** Back on the main branch, Elara (alone for now) enters the woods. Generate a paragraph describing her first steps into the Whispering Woods.
  + *LLM Prompt Example: "Elara, alone, steps into the Whispering Woods for the first time. Describe the atmosphere and her initial feelings. One paragraph."*
* **Git Actions (ensure you are on the** main **branch –** git branch **to check):**
  + Open chapter1.txt and append this new paragraph. Save.
  + Stage and commit this change to the main branch:
    - git add chapter1.txt
    - git commit -m "Chapter 1: Elara enters the Whispering Woods"
* **What you'll learn:** Making changes on main while another branch exists.

Task6:

* **Story Action:** You decide the companion idea is good and want to include Shadow in the main story.
* **Git Actions (ensure you are on the** main **branch):**
  1. Merge the add-companion branch into main: git merge add-companion
  2. Git will likely perform a "fast-forward" merge if characters.txt wasn't changed on main since the branch was created, or it will try to auto-merge. If it asks for a commit message for the merge, you can usually accept the default.
  3. Open characters.txt. Shadow's description should now be there!
  4. View the log to see the merge: git log --oneline --graph
* **What you'll learn:** merge, understanding how changes from branches are combined.

Task7:

* **Story Action:** You start writing a new section for Chapter 2, but it's terrible. You want to discard these *uncommitted* changes.
  + *LLM Prompt Example: "Generate a silly, out-of-place sentence that Elara might think in the woods, e.g., about wanting pizza."*
* **Git Actions:**
  + Create a new file chapter2.txt.
  + Add the silly sentence to chapter2.txt and save.
  + Stage it: git add chapter2.txt
  + Check status: git status (shows chapter2.txt staged for commit).
  + You realize it's bad. Unstage the file: git reset chapter2.txt
  + Check status: git status (shows chapter2.txt as untracked again).
  + Now, to discard the actual changes in your working directory (delete the content or the file):
    - Option A (discard changes in the file, keep the empty file): git checkout -- chapter2.txt
    - Option B (just delete chapter2.txt from your file system).
  + If you did Option A, check git status. It might show it as untracked or gone depending on how git checkout -- works on an empty new file vs. modified. If untracked, you can delete it. If you want to remove an untracked file known to Git previously: git clean -fd (Use with caution: -f is force, -d is directories). For a single file git clean -f chapter2.txt.
  + *Alternatively, simply delete* chapter2.txt *manually.*
* **What you'll learn:** reset (to unstage), checkout -- <file> (to discard working directory changes). git clean is a bit more advanced but good to know for untracked files.

Task8:

* **Story Action:** You like to keep personal notes or LLM prompts in a file called my\_story\_notes.txt, but you don't want Git to track this.
* **Git Actions:**
  1. Create a file named .gitignore (note the dot at the beginning).
  2. In .gitignore, add a new line with: my\_story\_notes.txt
  3. Save .gitignore.
  4. Stage and commit .gitignore:
     + git add .gitignore
     + git commit -m "Add .gitignore to exclude personal notes"
  5. Now, create my\_story\_notes.txt and put some text in it.
  6. Check status: git status. You should *not* see my\_story\_notes.txt listed, because it's being ignored.
* **What you'll learn:** .gitignore to exclude files/patterns from tracking.

Task 9:

* **Story Action:** You just made a commit for Chapter 1 but realized you had a typo in the commit *message* or forgot a small change in the file.
* **Git Actions:**
  1. Let's say your last commit was: git commit -m "Chapter 1: Elara enters teh Whispering Woods" (Note the typo "teh").
  2. Open chapter1.txt and fix a small typo *in the story content itself* or add one more sentence. Save the file.
  3. Stage the change: git add chapter1.txt
  4. Now, amend the previous commit (this will combine your new staged change and let you edit the old message):
  5. git commit --amend
  6. Your text editor will open with the old commit message. Correct "teh" to "the". Save and close the editor.
  7. Check the log: git log -1 (shows the last commit). The message should be fixed, and the file change included. The old commit is gone, replaced by a new one.
  8. **(Caution: Only amend commits that haven't been pushed to a shared repository).**
* **What you'll learn:** commit --amend to fix the last commit.

Task 10:

* **Story Action:**
  + On main branch: Generate a "happy ending" for your story (a short paragraph) and put it in a new file ending.txt. Commit it.
    - *LLM Prompt (main): "Write a short happy ending for 'The Whispering Woods Chronicle' where Elara solves the map's mystery and brings prosperity."*
  + Now, create a new branch: git branch tragic-ending
  + Switch to it: git checkout tragic-ending
  + On tragic-ending branch: *Modify* ending.txt to have a "tragic ending". Commit it on this branch.
    - *LLM Prompt (tragic-ending): "Rewrite the ending in 'ending.txt' to be a tragic one where Elara gets lost forever."*
* **Git Actions:**
  + **On** main **branch:**
    - Create ending.txt with happy ending.
    - git add ending.txt
    - git commit -m "Add happy ending"
  + git branch tragic-ending
  + git checkout tragic-ending
  + **On** tragic-ending **branch:**
    - Modify ending.txt with the tragic ending content.
    - git add ending.txt
    - git commit -m "Add tragic ending"
  + Switch back to main: git checkout main
  + Try to merge tragic-ending: git merge tragic-ending
  + **CONFLICT!** Git will tell you there's a merge conflict in ending.txt because both branches changed the same lines.
  + Open ending.txt. You'll see markers like <<<<<<< HEAD, =======, >>>>>>> tragic-ending.
  + **Your task:** Edit ending.txt to resolve the conflict. You can:
    - Keep the HEAD (main/happy) version.
    - Keep the tragic-ending version.
    - Write a new ending that combines elements of both or is entirely different.
    - Make sure to remove the <<<<<<<, =======, >>>>>>> markers.
  + Save ending.txt.
  + Stage the resolved file: git add ending.txt
  + Complete the merge with a commit: git commit -m "Resolve ending conflict, choosing [your choice]"
* **What you'll learn:** Experiencing and resolving a merge conflict – a crucial skill!

Task11:

* **Story Action:** You feel the story with its chosen ending is complete for a "first edition."
* **Git Actions:**
  1. Ensure you're on the main branch and all desired changes are committed.
  2. Create an annotated tag: git tag -a v1.0 -m "First complete version of The Whispering Woods Chronicle"
  3. View tags: git tag
  4. View info about a specific tag: git show v1.0
* **What you'll learn:** tag to mark specific important points in history.

Task12 :

This requires a second "remote" repository. For simplicity, we can make another local folder act as a remote.

* **Git Actions:**
  1. Outside your whispering-woods-chronicle project folder, create a new *empty* folder, e.g., whispering-woods-remote.git.
  2. Navigate into whispering-woods-remote.git and initialize a "bare" repository (this acts like a central server):
  3. cd ../whispering-woods-remote.git (or the correct path)
  4. git init --bare
  5. Go back to your whispering-woods-chronicle project directory:
  6. cd ../whispering-woods-chronicle
  7. Add this "remote" to your project:
  8. git remote add origin ../whispering-woods-remote.git (use the correct path to your bare repo)
  9. Verify the remote: git remote -v
  10. Push your main branch to the origin remote:
  11. git push -u origin main
  12. Push your tags: git push --tags
* **Story Action:** You've now "published" your story to a central place where a collaborator (or your future self on another computer) could git clone ../whispering-woods-remote.git to get a copy.
* **What you'll learn:** init --bare, remote add, push. (And conceptually, clone and pull for the other side).

Task 13:

* **Story Action:** You're working on a new side-quest branch. You generate a brilliant idea for a magical item. You *think* you're on side-quest, write about the item in magical\_items.txt, commit it, but then realize you accidentally committed it to main!
  + *LLM Prompt: "Describe a magical amulet called the 'Orb of Fleeting Moments' that allows brief glimpses into the past. 50 words."*
* **Git Actions:**
  + Ensure you are on the main branch.
  + Create magical\_items.txt with the LLM content.
  + git add magical\_items.txt
  + git commit -m "Introduce Orb of Fleeting Moments"
  + *(Realization)* "Oh no, that was for the side-quest!"
  + Create and switch to the side-quest branch if it doesn't exist:
  + git branch side-quest
  + git checkout side-quest
  + Now, on side-quest, you want that commit. Cherry-pick it from main:
  + git cherry-pick main (This assumes the commit you want is the latest on main. If not, you'd use the commit hash from main's log.)
  + Verify magical\_items.txt and the commit are on side-quest.
  + Switch back to main: git checkout main
  + Remove the mistaken commit from main. Since it was the *last* commit and hasn't been pushed:
  + git reset --hard HEAD~1 (Caution: --hard discards changes. Ensure this is the *only* thing you want to undo from that commit on main).
  + Verify on main that magical\_items.txt is gone/empty and the commit is removed from main's log, but it's still safe on side-quest.
* **What you'll learn:** cherry-pick to grab specific commits, reset --hard HEAD~1 to undo the last local commit on a branch. (Emphasize caution with reset --hard).

Task14:

* **Story Action:** You were rapidly brainstorming plot points for Chapter 3. You made three quick, messy commits: "Add vague villain idea", "Elara finds clue A", "Maybe a talking squirrel?". Now you want to clean this up into *one* good commit before sharing.
  + *LLM Prompt 1: "Villain concept: a shadowy sorcerer who feeds on forgotten memories. 20 words."*
  + *LLM Prompt 2: "Clue for Elara: she finds a scorched piece of parchment with half an ancient rune. 20 words."*
  + *LLM Prompt 3: "Silly idea for Chronomancer's Cipher: Elara meets a time-traveling squirrel named Squeaky. 20 words."*
* **Git Actions:**
  + Create chapter3\_ideas.txt.
  + Commit 1: Add villain concept. git add . && git commit -m "WIP: villain draft"
  + Commit 2: Add clue concept to the same file. git add . && git commit -m "WIP: clue for Elara"
  + Commit 3: Add squirrel idea to the same file. git add . && git commit -m "WIP: talking squirrel?"
  + Now, use interactive rebase to squash these: git rebase -i HEAD~3
  + Your editor will open. Change pick to s (or squash) for the second and third commits. Save and close.
  + Another editor window opens for the new combined commit message. Edit it to something like "Consolidate initial plot ideas for Chapter 3" (and you can refine the body to include the actual text ideas if you like). Save and close.
  + Check git log -1 (or git log --oneline -3). You should see one new clean commit instead of three messy ones.
* **What you'll learn:** rebase -i for squashing commits (cleaning up local history).

Task 15:

* **Story Action:** You had a fantastic idea for a subplot involving a hidden library. You wrote it down in hidden\_library.txt, committed it. Then, distracted, you accidentally did a git reset --hard HEAD~1 thinking you were undoing something else. The commit and file are gone!
  + *LLM Prompt: "Describe a hidden, ancient library accessible only through a riddle in 'The Chronomancer's Cipher'. 50 words."*
* **Git Actions:**
  + Create hidden\_library.txt with the LLM content.
  + git add hidden\_library.txt && git commit -m "Introduce hidden library subplot"
  + *(Accidentally)* git reset --hard HEAD~1
  + *(Panic)* The file is gone from the working directory! Check git log. The commit is gone!
  + The magic recovery tool: git reflog
  + Look through the reflog. You'll see your "Introduce hidden library subplot" commit and the reset. Identify the commit hash of your lost commit (e.g., abcdef0).
  + To get it back: You can either reset to it: git reset --hard abcdef0 (if you're sure you want to go back to that state entirely).
  + OR, a safer way if you just want that commit's changes back on top of your current work (if any):
    - Create a temporary branch at that lost commit: git branch recovered-idea abcdef0
    - Then you can cherry-pick it, merge it, or just inspect it.
    - For this exercise, let's just restore it: git reset --hard abcdef0
  + Verify hidden\_library.txt and the commit are back.
* **What you'll learn:** reflog as a safety net, recovering from a reset --hard by finding the commit hash.

Task 16:

* **Story Action:** You're halfway through writing a new scene in chapter4.txt (it's not ready to commit). Suddenly, your "editor" (a friend) messages you: "Urgent: there's a typo in chapter1.txt from the very first paragraph! Fix it ASAP!" You need to switch tasks without committing your messy chapter4.txt.
  + *LLM Prompt (for Chapter 4 in-progress): "Elara cautiously enters a crumbling tower, sensing ancient magic. Half-finished sentence: The air grew cold and..."*
* **Git Actions:**
  + Open chapter1.txt, introduce a typo if one isn't there, save. (Or just pretend it was there).
  + Create chapter4.txt and add the half-finished scene. Save it.
  + Check git status. chapter4.txt is modified and untracked (or just modified if it existed).
  + Stash your current changes: git stash push -m "WIP Chapter 4 scene" (The -m is good practice).
  + Check git status. Your working directory should be clean regarding chapter4.txt.
  + Fix the "typo" in chapter1.txt.
  + git add chapter1.txt && git commit -m "Fix typo in Chapter 1 opening"
  + Now, bring back your unfinished Chapter 4 work: git stash pop
  + (If there are conflicts, Git will tell you. For this simple case, there shouldn't be).
  + Verify chapter4.txt has your unfinished work back. You can see your stashes with git stash list.
* **What you'll learn:** stash push, stash pop, stash list for saving and restoring uncommitted changes temporarily.

Task 17 :

* **Story Action:** You wrote a long passage in world\_lore.txt that describes both the "Ancient Runes" and the "Mythical Creatures" of your world. You committed it all as "Added world lore". Now you realize it should have been two separate commits for better organization.
  + *LLM Prompt: "Describe two distinct lore elements for 'The Chronomancer's Cipher': 1. The language of Ancient Runes (50 words). 2. The Griffin, a mythical creature guarding the Sky Peaks (50 words)."*
* **Git Actions:**
  + Create world\_lore.txt. Paste *both* descriptions into it.
  + git add world\_lore.txt && git commit -m "Added world lore"
  + Reset the last commit but keep the changes in your working directory: git reset HEAD~1 (This is a "mixed" reset, the default. Changes are unstaged).
  + git status will show world\_lore.txt as modified.
  + Now, stage only parts of the file for the first commit. This is tricky via command line alone without opening an editor that supports partial staging. A common way:
  + git add -p world\_lore.txt (patch mode)
  + Git will show you chunks of changes. For each chunk:
    - y - stage this hunk
    - n - do not stage this hunk
    - s - split the current hunk into smaller hunks (if possible)
    - e - manually edit the current hunk (advanced)
    - q - quit
    - Selectively stage only the "Ancient Runes" part.
  + git commit -m "Document Ancient Runes lore"
  + Now, the rest of the changes (Mythical Creatures) are still in world\_lore.txt but unstaged or modified.
  + git add world\_lore.txt (to stage the remaining changes)
  + git commit -m "Describe Mythical Griffin"
  + Check git log -2 --oneline. You should see your two new commits.
* **What you'll learn:** reset HEAD~1 (mixed reset), git add -p (patch mode staging). This is a very useful skill.

Task 18:

* **Story Action:** You want to go back and read how Chapter 1 looked *before* you added the "Elara enters the woods" part, just to remember your initial pacing, without affecting your current story.
* **Git Actions:**
  1. Assume your history has:
     + Commit C: "Chapter 1: Elara enters the Whispering Woods" (HEAD)
     + Commit B: "Expand Chapter 1 with Elara's decision"
     + Commit A: "Initial draft of Chapter 1 opening"
  2. Find the hash of Commit B (e.g., from git log --oneline). Let's say it's aabbcc1.
  3. Checkout that commit directly: git checkout aabbcc1
  4. Git will warn you that you are in a 'detached HEAD' state. This is okay for temporary exploration.
  5. Open chapter1.txt. It will be the version from Commit B.
  6. **Story Change (Hypothetical):** While in this detached state, you have an idea for an *alternate* way Elara could have decided to enter the woods. You modify chapter1.txt with this new idea and save.
  7. *LLM Prompt for this: "An alternate way Elara decides to enter the woods: a cryptic message appears on the map only when held under moonlight."*
  8. git add chapter1.txt
  9. git commit -m "ALT: Elara deciphers map by moonlight"
  10. You've now made a commit from a detached HEAD. To save this alternate idea, create a branch *now*:
  11. git branch alternate-decision
  12. Switch back to your main work: git checkout main
  13. Your main branch is untouched. The alternate-decision branch exists with that experimental change.
* **What you'll learn:** Checking out old commits (detached HEAD), making experimental commits, and creating a new branch from a detached HEAD to save that work.