**Comprehensive Guide to Git Branching, Switching, and Merging #githubclass**

**Introduction**

Git is a powerful version control system used by developers worldwide to manage changes in code efficiently. Branching in Git allows multiple versions of the codebase to coexist, enabling developers to work on new features, bug fixes, or experiments without affecting the main production code.

**Understanding Git Branching and Working Directory**

**Initial Branch Setup and Divergence**

* When you start a Git project, the default branch is often called main (or sometimes master).
* You can create additional branches such as development for ongoing work. Initially, the branches are identical because they share the same commit history.
* As development progresses, changes in main and development diverge because each branch evolves independently with new commits.

**Single Folder, Multiple Branches**

* In Git, you work within a single folder (called the working directory) regardless of how many branches exist.
* When you switch branches using Git commands or GUI tools, Git updates the files in this folder to reflect the version saved in that branch.
* This means you do not have multiple physical copies of the project; instead, Git dynamically updates files based on the branch you checked out.

**Switching Branches and Handling Changes**

**Switching Between Branches**

* Switching branches changes the contents of files in your working directory to those that belong to the branch you switch to.
* This switch happens instantly but you remain in the same folder on your machine.

**Uncommitted Changes and Conflicts on Switch**

* If you have made changes in your current branch and haven’t committed them yet, switching branches can cause conflicts if those files are different in the branch you want to switch to.
* Git will warn you that you have uncommitted changes and may prevent the switch to avoid data loss.

**Stashing Changes**

* Git provides a command called git stash that temporarily saves your uncommitted changes.
* When you stash changes, Git cleans your working directory so you can switch branches safely.
* You can later restore (or "apply") these changes back onto any branch using git stash apply or git stash pop.

**Committing and Pushing Changes**

**Local Commits**

* Changes to files are first saved (staged) and then committed in your local Git repository.
* Committing records the snapshot of your files in the current branch.

**Pushing to Remote Repository**

* To share your changes with others, you push your local commits to the remote Git repository (e.g., GitHub, GitLab).
* Branches that do not yet exist on the remote can be pushed for the first time using git push -u origin branch-name.

**Feature Branch Workflow and Collaboration**

**Creating Feature Branches**

* Instead of directly working on main or development, create a new branch from development for your feature or bug fix.
* This isolates your changes and makes managing multiple tasks easier.

**Publishing and Sharing Feature Branches**

* Push your feature branch to the remote repo so others can see and collaborate.

**Pull Requests (PR) / Merge Requests (MR)**

* Once your feature is ready, open a Pull Request to propose merging your feature branch back into the development branch.
* The PR includes a description of changes and requests code review from teammates.

**Code Review and Merge Hooks**

* Teams usually require PRs to be reviewed and approved before merging.
* Protected branches like main or development often have branch protection rules that prevent direct commits and enforce PR reviews.

**Merging and Resolving Conflicts**

**Merging Branches**

* After approval, merge the feature branch into development. Later, development can be merged into main for production release.
* Merging integrates all commits from the feature branch into the target branch.

**Merge Conflicts**

* Conflicts happen when the same file is changed in conflicting ways in two branches.
* Git flags these conflicts and requires manual intervention to resolve by choosing which changes to keep or combining both.

**Best Practices in Branching and Merging**

* Always create feature branches instead of working directly on main or development.
* Commit your changes frequently with meaningful messages.
* Use stash to save work temporarily without committing.
* Keep your branches updated by regularly merging development or main changes into your feature branches.
* Use Pull Requests for code review and collaboration.
* Resolve merge conflicts carefully and test your code after merging.
* Use branch protection rules to prevent accidental direct commits to critical branches.

**15 Important Points to Remember**

1. main and development branches start the same but diverge over time.
2. You work in a single folder; switching branches changes the file contents dynamically.
3. Uncommitted changes can block branch switching if they conflict with target branch files.
4. git stash safely saves uncommitted changes so you can switch branches without losing work.
5. Commits save your changes locally in the branch you are working on.
6. git push uploads your commits to the remote repository.
7. Feature branches isolate new development work from main and development.
8. Push feature branches to remote for collaboration and pull requests.
9. Pull Requests enable code review before merging changes into development or main.
10. Protected branches prevent direct commits and require PR approval to merge.
11. Merging integrates changes from one branch into another.
12. Merge conflicts require manual resolution before the merge can complete.
13. Regularly update feature branches from development to minimize conflicts.
14. Clear commit messages and frequent commits improve collaboration.
15. Follow branching workflows and use PRs to maintain code quality and project stability.

**10 Common Interview Questions with Detailed Answers**

**1. What is the difference between the main branch and a development branch?**

**Answer:**  
main is usually the stable branch that holds production-ready code. The development branch is used for integrating ongoing development work and testing before changes are merged into main. This helps keep production code stable while development continues separately.

**2. How does Git handle switching between branches in the same directory?**

**Answer:**  
Git updates the files in your working directory to reflect the snapshot of the selected branch. You don’t get multiple copies of the repository; instead, Git replaces the files dynamically, so your folder’s content corresponds to the checked-out branch.

**3. What happens if you try to switch branches with uncommitted changes?**

**Answer:**  
Git will check if the uncommitted changes conflict with files in the target branch. If yes, Git prevents switching to avoid losing those changes. If no conflicts exist, Git lets you switch and retains the changes.

**4. What is git stash and when would you use it?**

**Answer:**  
git stash temporarily saves your uncommitted changes and cleans your working directory, allowing you to switch branches or pull new changes safely. You can later reapply these changes on any branch.

**5. Explain the feature branch workflow in Git.**

**Answer:**  
Developers create separate branches for each feature or bug fix from the development branch. They work independently, commit locally, push to remote, and create pull requests to merge changes back into development. This isolates work, enables parallel development, and helps maintain stable branches.

**6. How do you push a new branch to a remote repository?**

**Answer:**  
After creating a branch locally and committing changes, use:  
git push -u origin branch-name  
This command pushes the branch to the remote repository and sets the upstream tracking reference.

**7. What is a Pull Request and why is it important?**

**Answer:**  
A Pull Request (PR) is a request to merge changes from one branch into another. It enables code review, discussions, and automated checks before merging, ensuring higher code quality and collaboration.

**8. How do you resolve a merge conflict in Git?**

**Answer:**  
When a conflict occurs, Git marks the conflicting sections in the affected files. You manually edit these files to resolve conflicts by choosing which changes to keep or combining both, then stage and commit the resolved files.

**9. What are some best practices to follow when working with Git branches?**

**Answer:**

* Work on feature branches, not directly on main.
* Commit frequently with clear messages.
* Use pull requests for review and integration.
* Keep your branches updated by regularly merging from base branches.
* Resolve conflicts carefully and test after merges.

**10. How do you keep your feature branch updated with changes from the main branch?**

**Answer:**  
You can merge main into your feature branch using:  
git merge main  
or rebase your feature branch on top of main using:  
git rebase main  
This incorporates the latest changes and reduces conflicts during the final merge.