# Git Commands and Best Practices Week -6

## 1. Stage all changes and commit with a meaningful commit message

To stage \*\*all\*\* modified, deleted, and new files and commit them:  
  
```  
git add .  
git commit -m "Add meaningful message describing what was changed"  
```  
  
\*\*Explanation:\*\*  
- `git add .` stages all changes in the current directory and subdirectories.  
- `git commit -m "..."` creates a commit with the provided message, explaining the purpose of the changes for future reference.

## 2. Move commits to the correct branch (committed on the wrong branch)

If you made commits on the wrong branch (e.g., `wrong-branch`) and want them on `correct-branch`:  
  
```  
# Make sure you are on the wrong branch with the commits  
git log # verify the commits  
  
# Create and switch to the correct branch based on where you should have committed  
git checkout correct-branch  
  
# Merge the changes from the wrong branch (keep commits history)  
git cherry-pick <commit-hash> # Or multiple cherry-pick commands  
  
# Alternatively, move all recent commits (e.g., last 3 commits) to the correct branch  
git checkout correct-branch  
git cherry-pick wrong-branch~3..wrong-branch  
  
# Now remove the commits from the wrong branch  
git checkout wrong-branch  
git reset --hard HEAD~3  
```  
  
\*\*Explanation:\*\*  
- `git cherry-pick` applies specific commits to another branch.  
- `git reset --hard HEAD~3` removes the last 3 commits from the wrong branch (be cautious if working collaboratively).

## 3. Create a new branch, make changes, and push to remote

```  
# Create and switch to a new branch  
git checkout -b feature-branch-name  
  
# Make your code changes, then stage and commit  
git add .  
git commit -m "Implement feature X"  
  
# Push the branch to remote repository (e.g., origin)  
git push -u origin feature-branch-name  
```  
  
\*\*Explanation:\*\*  
- `git checkout -b` creates and immediately switches to the new branch.  
- `-u` sets the upstream branch, making future `git push` and `git pull` simpler.

## 4. Contribute to an open-source project on GitHub

\*\*Step-by-step:\*\*  
  
1. \*\*Fork the repository:\*\*  
 - Go to the project’s GitHub page.  
 - Click on the \*\*Fork\*\* button to create a personal copy under your GitHub account.  
  
2. \*\*Clone your fork:\*\*  
```  
git clone https://github.com/your-username/repo-name.git  
cd repo-name  
```  
  
3. \*\*Add the original project as upstream:\*\*  
```  
git remote add upstream https://github.com/original-author/repo-name.git  
```  
  
4. \*\*Create a new branch:\*\*  
```  
git checkout -b feature-branch  
```  
  
5. \*\*Make changes, stage, and commit:\*\*  
```  
git add .  
git commit -m "Describe your changes"  
```  
  
6. \*\*Push the branch to your fork:\*\*  
```  
git push -u origin feature-branch  
```  
  
7. \*\*Create a Pull Request (PR):\*\*  
 - Go to your fork on GitHub.  
 - Click \*\*Compare & pull request\*\*.  
 - Provide a clear title and description, then submit the PR.

## 5. Resolve merge conflicts

\*\*When merging your branch into the main branch:\*\*  
  
```  
# Make sure you are on your feature branch  
git checkout feature-branch  
  
# Fetch the latest changes  
git fetch origin  
  
# Merge the main branch into your branch  
git merge origin/main  
```  
  
If there are conflicts:  
- Open conflicting files; you will see conflict markers (`<<<<<<<`, `=======`, `>>>>>>>`).  
- Edit to resolve conflicts.  
- After resolving, mark the conflicts as resolved and commit:  
  
```  
git add .  
git commit -m "Resolve merge conflicts between feature-branch and main"  
```  
  
\*\*Explanation:\*\*  
- `git merge` brings in changes from another branch.  
- Conflicts must be manually resolved before completing the merge.

## 6. Create a feature branch based on the latest main branch

```  
# Fetch the latest data from the remote  
git fetch origin  
  
# Switch to main branch and ensure it's up to date  
git checkout main  
git pull origin main  
  
# Create and switch to the new feature branch  
git checkout -b new-feature-branch  
```  
  
\*\*Explanation:\*\*  
- This ensures your new branch starts from the latest code on the main branch, avoiding outdated changes.

## 7. Revert to a specific commit, discarding later commits

```  
# Hard reset to the specific commit  
git reset --hard <commit-hash>  
```  
  
Then, force push if the branch is shared:  
  
```  
git push --force  
```  
  
\*\*Explanation:\*\*  
- `git reset --hard` moves the HEAD and branch pointer to the chosen commit, deleting all later commits.  
- Force push is needed if changes were already pushed to the remote.  
  
\*\*⚠️ Caution:\*\* This rewrites history; coordinate with your team before using.

## 8. Restore a deleted file from previous commit

If you deleted `file.txt` and committed:  
  
```  
# Restore file from the previous commit (HEAD~1)  
git checkout HEAD~1 -- file.txt  
  
# Stage and commit the restoration  
git add file.txt  
git commit -m "Restore accidentally deleted file.txt"  
```  
  
\*\*Explanation:\*\*  
- `git checkout <commit> -- <file>` recovers a file as it existed in that commit.  
- You then commit to keep the file restored in your branch.