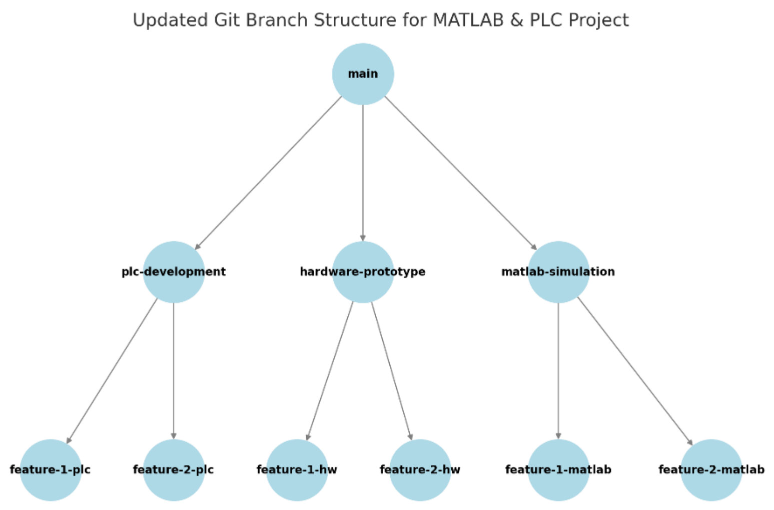
Comprehensive Git Cheat Sheet



Git Best Practices

- Make small, frequent commits to track progress effectively.
- Write concise, yet descriptive commit messages.
- Use branches for features and bug fixes to keep 'main' stable.
- Always pull the latest changes before starting work.
- Review changes using 'git diff' before committing.
- Use '.gitignore' to avoid committing unnecessary files.
- Merge frequently to minimize conflicts.
- Use Git LFS for large binary files like Omron PLC and MATLAB files.
- Limit commits to a single type of change (e.g., bug fix, new feature, refactoring).
- Format commit messages properly

Example Commit Message

feat: Added motor safety interlock in PLC

Implemented a new safety interlock to prevent motor activation

when the emergency stop button is pressed.

- Updated ladder logic in main program
- Added interlock conditions for safety relay
- Tested successfully in simulation mode

Setting Up .gitignore & Handling Untracked Files

1. Create a .gitignore file in the root of your repository:

```
touch .gitignore
```

2. Add file patterns to exclude specific files and directories:

```
# Ignore compiled files
*.o
*.out
*.exe
# Ignore logs and temp files
        *.log
        *.tmp
# Ignore system files
        .DS_Store
        Thumbs.db
```

3. Ignore files globally across all repositories:

```
git config --global core.excludesfile ~/.gitignore_global
echo '*.log' >> ~/.gitignore_global
```

4. Track empty directories using a placeholder file (.gitkeep):

```
mkdir logs
touch logs/.gitkeep
git add logs/.gitkeep
git commit -m 'Added .gitkeep to track empty logs directory'
```

5. View and clean untracked files:

```
git status
git clean -n # Show what will be deleted
git clean -f # Remove untracked files
```

Git Ignore Pattern Glossary

- `/` : Indicates a directory. Example: `logs/` ignores the logs directory.

- `*` : Matches any number of characters. Example: `*.log` ignores all `.log` files.

- `**` : Matches multiple directories. Example: `logs/**` ignores all files inside `logs/`.

- `?` : Matches a single character. Example: `file?.txt` matches `file1.txt`, `fileA.txt`, etc.

- `!` : Negates a pattern. Example: `!important.log` ensures `important.log` is **not ignored**.

Merging Development Branches into Main

Ensure all changes are committed in each branch:

```
git checkout plc-development

git commit -am 'Finalizing changes before merge'
```

Switch to main and pull the latest version:

```
git checkout main

git pull origin main
```

Merge each branch into main:

```
git merge plc-development

git merge matlab-simulation

git merge hardware-prototype
```

Resolve any merge conflicts if needed:

```
git status

git add .

git commit -m 'Resolved merge conflicts'
```

Push the updated main branch:

```
git push origin main
```

Handling Pull Request Merging Permissions on GitHub

To restrict who can merge PRs, go to:

Settings → Branches → Add branch protection rules

Enable:

Require pull request reviews before merging

Restrict who can push to the branch (Only maintainers/admins)

Require status checks to pass before merging

Use CODEOWNERS to auto-assign reviewers based on files.

Rebasing vs. Merging When Syncing from main

Option 1: Merge (Simple)

```
git checkout plc-development
git merge main
git push origin plc-development
```

Option 2: Rebase (Cleaner History)

```
git checkout plc-development
git rebase main
```

Resolve conflicts if necessary:

```
git add .
git rebase --continue
git push origin plc-development --force
```

Repository Setup & Configuration

- Initialize a new repo: git init
- Clone an existing repo: git clone <repo-url>

Branch Management

- Create a new branch: git checkout -b <branch-name>
- Switch branches: git checkout <branch-name>
- List all branches: git branch
- Delete a branch: git branch -d <branch-name>

Committing & Saving Changes

- Check status: git status
- Stage a file: git add <file-name>
- Stage all changes: git add .
- Commit changes: git commit -m "Commit message"
- Show differences before committing: git diff
- Review staged changes: git diff --staged
- Show word-level changes: git diff --word-diff
- Show colored diff output: git diff --color

Viewing Commit History

- View full commit history: git log
- View compact commit history: git log --oneline

Pushing & Syncing with GitHub

- Pull latest changes: git pull origin main
- Push changes: git push origin <branch-name>

Merging & Managing Code

- Merge a branch: git merge <branch-name>
- Fix merge conflicts:
  git add .
  git commit -m "Resolved conflict"

Undo & Fix Mistakes

- Undo last commit (keep changes unstaged): git reset HEAD~1
- Undo last commit (discard changes): git reset --hard HEAD~1
- Revert a pushed commit: git revert <commit-hash>
- Stash changes: git stash
- Apply stashed changes: git stash pop
- Restore an unstaged file: git restore <file-name>
- Unstage a file (keep changes): git restore --staged <file-name>

Moving & Renaming Files

- Rename a file: git mv old_filename new_filename
- Move a file: git mv filename new_directory/

Working with Git LFS

- Track large files: git lfs track '*.cxp' '*.smc2' '*.mat'
- Check LFS files: git lfs ls-files
- Pull LFS files: git lfs pull