Git & GitHub Cheat Sheet

AppTrainers

General Commands

- `cd` : Change directory
- `ls` : List directory
- `pwd` : Print working directory
- `touch` : Create a file
- `mkdir` : Create a directory
- `rm` : Remove a file
- `rmdir` : Remove a directory
- `cp` : Copy a file
- `mv` : Move or rename a file/directory
- `cat` : Concatenate and display the content of a file
- `tree` : Display the directory structure in a tree format

What is Git?

- A version control system that tracks changes in code over time.

Why Use Git?

- Enables collaboration, tracks changes, and allows reverting to previous versions.

Installing Git

- Check if installed: `git --version`
- **From Official Website**: https://git-scm.com/
- **Using Package Managers**:
  - Windows: `winget install -e --id Git.Git`
  - macOS: `brew install git`
  - Linux: `sudo apt-get install git`

Configuring Git

Levels of Configuration:

1. **System-Level**: `--system`
2. **Global-Level**: `--global`
3. **Local-Level**: `--local`

Commands:

- Set name: `git config --global user.name "Your Name"`
- Set email: `git config --global user.email "your.email@example.com"`
- List configurations: `git config --list`
- Set default editor: `git config --global core.editor "code --wait"`
- Edit global config: `git config --global --edit`
- Set default branch: `git config --global init.defaultBranch main`

Git Initialization

- Initialize repository: `git init`
- Uninitialize repository: `git rm .git/ -rf`

Git Status

- Show status: `git status`

Git Add

- Add specific file: `git add <file>`
- Add all files: `git add .` (Not recommended)
- Untrack a file: `git rm --cached <file>`
- Remove from staging: `git reset <file>` or `git restore --staged <file>`

Git Ignore

- Create `.gitignore` file in root directory.

- Add files/directories to ignore.

Git Commit

- Commit with message: `git commit -m "Commit message"`
- Commit with description: `git commit -m "Subject" -m "Description"`
- Amend last commit: `git commit --amend`

The Perfect Commit

1. Use `git status` to see changes.
2. Use `git add` to stage changes selectively.
3. Use `git diff` to review staged changes.
4. Use `git add -p` for interactive staging.

The Perfect Commit Message

- Subject: Concise summary (<80 characters).
- Body: Detailed explanation (what changed, why, and what to watch out for).

Example: `feat: add new feature`

Git Log

- Show commit history: `git log`
- Useful flags:
    - `--oneline`: Condensed history
    - `--graph`: Graphical history
    - `--all`: Show all branches

Git Alias

- Create alias: `git config --global alias.<alias-name> "<command>"`
- Example: `git config --global alias.lg "log --oneline --graph --all"`

Git Remote

- Add remote: `git remote add origin <url>`
- Change remote URL: `git remote set-url origin <new-url>`
- List remotes: `git remote -v`
- Remove remote: `git remote remove origin`

Git Push

- Push to remote: `git push -u origin main`
- Force push: `git push -f`

Git Pull

- Pull changes: `git pull origin main`
- Fetch changes: `git fetch origin main`
- Merge changes: `git merge origin/main`

Git Clone

- Clone repository: `git clone <url>`
- Clone to specific folder: `git clone <url> <folder>`

GitHub Codespaces

- Create a codespace via GitHub repository's `Code` button.

Git Branches

- List branches: `git branch`
- Create branch: `git branch <branch-name>`
- Delete branch: `git branch -d <branch-name>`

- Force delete: `git branch -D <branch-name>`
- Rename branch: `git branch -M <new-name>`

Git Checkout

- Switch branch: `git checkout <branch-name>`
- Create and switch: `git checkout -b <branch-name>`
- Discard changes: `git checkout -- <file>` or `git restore <file>`

Git Merge

- Merge branches: `git merge <branch-name>`
- Abort merge: `git merge --abort`

Merge Conflicts

- Resolve conflicts manually by editing files.
- Markers: `<<<<<<<`, `=======`, `>>>>>>>`
- Continue merge: `git add .` then `git merge --continue`

Feature Branch Workflow

1. Create branch: `git checkout -b feature-branch-name`
2. Develop feature and commit changes.
3. Push branch: `git push origin feature-branch-name`
4. Create pull request on GitHub.
5. Code review and merge.

Branching Strategies

- **GitHub Flow**: One long-running branch (`main`) + short-lived branches.
- **Git Flow**: Includes `main`, `develop`, `feature`, `release`, and `hotfix` branches.

Pull Requests

- Fork repository, clone, create branch, make changes, push, and open pull request.

Merge vs Rebase

- **Merge**: Creates a new merge commit.
- **Rebase**: Rewrites history to make it linear.

Stash

- Save changes: `git stash`
- List stashes: `git stash list`
- Apply last stash: `git stash apply`
- Remove last stash: `git stash drop`
- Clear all stashes: `git stash clear`

Interactive Rebase

- Go back `n` commits: `git rebase -i HEAD~n`
- Actions:
    - `pick`: Keep commit as is.
    - `reword`: Change commit message.
    - `squash`: Combine with previous commit.

Cherry-Pick

- Copy commit: `git cherry-pick <commit-hash>`

Reflog

- View history: `git reflog`
- Recover lost branch: `git checkout -b <branch-name> <commit-hash>`

Revert

- Undo commit: `git revert <commit-hash>`
- Undo last commit: `git revert HEAD`

Submodules

- Add submodule: `git submodule add <repository-url>`

Search and Find

- By date: `git log --after="YYYY-MM-DD" --before="YYYY-MM-DD"`
- By message: `git log --grep="keyword"`
- By author: `git log --author="name"`
- By file: `git log -- <file>`
- By branch differences: `git log branch1..branch2`