Complete Notes on Git and GitHub

1. Introduction to Version Control Systems (VCS)

A Version Control System (VCS) is software that helps developers manage changes to source code over time. It allows multiple developers to collaborate on a project and track changes.

- Types of Version Control Systems:
  - Local VCS: Stores changes on the local machine.
  - Centralized VCS (CVCS): Stores changes on a central server. Examples include SVN, CVS.
  - Distributed VCS (DVCS): Stores changes on both the central server and local machine. Examples include Git, Mercurial.

2. What is Git?

Git is a Distributed Version Control System. It tracks changes to files, allowing multiple developers to work on a project concurrently. It maintains both local and remote repositories.

- Advantages of Git:
  - Distributed
  - Fast and efficient
  - Branching and merging
  - Strong support for non-linear development (branching)

3. Git vs. GitHub

- Git: A VCS used to track changes in source code.
- GitHub: A web-based platform that uses Git and provides features like code hosting, pull requests, issue tracking, and more.

4. Basic Git Terminology

- Repository (Repo): A directory that contains your project files and tracks changes. A repository can be local (on your machine) or remote (on GitHub, GitLab, etc.).
- Commit: A record of changes made to the files in the repository. It's like saving your work with a message describing what was done.
- Branch: A separate workspace to develop new features or fix bugs without affecting the main codebase.
- Merge: Integrating changes from one branch into another.
- Clone: Creating a local copy of a remote repository.
- Pull: Fetching the latest changes from the remote repository and merging them into your local branch.
- Push: Sending your local commits to a remote repository.
- Staging Area (Index): A place where changes are added before committing.
- Remote Repository: A version of your project hosted on the internet or another network.

5. Installing Git

To use Git locally, you need to install it on your system:

- On Linux: sudo apt install git
- On macOS: brew install git
- On Windows: Use the Git installer from git-scm.com.

Configuration

After installation, configure Git with your name and email:

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

6. Basic Git Commands

a. Initializing a Git Repository

To start tracking a project:

```
git init
```

b. Cloning a Repository

To copy a remote repository to your local system:

```
git clone <repository_url>
```

c. Checking the Status

Check the status of your working directory:

```
git status
```

d. Staging Changes

Add specific files to the staging area:

```
git add <filename>
```

Add all changes to the staging area:

```
git add .
```

e. Committing Changes

Create a commit with a message:

```
git commit -m "Commit message"
```

f. Viewing Commit History

To see the commit history:

```
git log
```

g. Pushing Changes

Push local commits to a remote repository:

```
git push origin <branch_name>
```

h. Pulling Changes

Fetch and merge changes from a remote repository:

```
git pull origin <branch_name>
```

7. Working with Branches

a. Creating a New Branch

To create a new branch:

```
git branch <branch_name>
```

b. Switching Between Branches

To switch to an existing branch:

```
git checkout <branch_name>
```

c. Merging Branches

To merge another branch into your current branch:

```
git merge <branch_name>
```

d. Deleting a Branch

Delete a branch once it's merged:

```
git branch -d <branch_name>
```

8. Undoing Changes

a. Undo Unstaged Changes

To discard changes in the working directory:

```
git checkout -- <filename>
```

b. Unstage Changes

To remove changes from the staging area:

```
git reset <filename>
```

c. Undo a Commit

Undo the last commit, but keep changes unstaged:

```
git reset --soft HEAD^
```

9. Collaborating on GitHub

a. Forking a Repository

To create your copy of someone else's repository:

- Go to the repository on GitHub and click the "Fork" button.

b. Creating Pull Requests (PR)

After making changes in your fork, you can create a PR to the original repository:

1. Go to your forked repository.
2. Click on "New Pull Request".
3. Add a description and click "Create Pull Request".

c. Issues and Bug Tracking

GitHub provides a way to track issues and feature requests:

1. Navigate to the "Issues" tab.

2. Create a new issue and describe the problem or request.

10. Advanced Git Commands

a. Stashing Changes

Save your uncommitted changes temporarily:

```
git stash
```

To retrieve stashed changes:

```
git stash pop
```

b. Rebasing

Rebasing re-applies commits on top of another branch:

```
git rebase <branch_name>
```

c. Tagging

Create a tag for marking specific commits (e.g., version releases):

```
git tag <tag_name>
```

Push tags to a remote repository:

```
git push --tags
```

11. GitHub Actions

GitHub Actions automates workflows like testing and deploying code. You define these actions in a .yml file within the .github/workflows/ directory.

Example:

```
name: CI

on: [push]

jobs:

  build:

    runs-on: ubuntu-latest

    steps:

      - uses: actions/checkout@v2

      - name: Run a script

        run: echo Hello, world!
```

12. Git Best Practices

- Commit frequently with descriptive messages.
- Use branches for new features and bug fixes.
- Pull before pushing to avoid conflicts.
- Write clear PR descriptions and link related issues.
- Review changes before merging PRs.

13. Common Git Workflows

a. Feature Branch Workflow

1. Create a new branch for a feature.
2. Work on the feature, then push it.
3. Open a PR to the main branch.
4. After approval, merge the feature branch.

b. Gitflow Workflow

1. Have a master branch and a develop branch.

2. Feature branches are created from develop.

3. Once features are complete, they are merged into develop for integration.

4. master is updated only after stable releases.