Complete Notes on Git and Version Control

1. Introduction to Git and Version Control

What is Version Control?

Version control is a system that records changes to files over time so you can recall specific versions later.
Helps multiple people collaborate on projects without overwriting each other's work.
Enables tracking, comparing, and reverting changes if needed.

Why Use Git?

Git is a popular distributed version control system.
Enables local and remote repository management.
Efficient handling of large projects.
Facilitates branching and merging workflows.
Widely supported and integrates with platforms like GitHub, GitLab, Bitbucket.

Basic Git Workflow Overview

Initialize a repo (git init)
Make changes in working directory
Stage changes (git add)
Commit changes (git commit)
Push commits to remote (git push)
Pull changes from remote (git pull)

2. Git Installation and Setup

Installing Git

Windows: Download from https://git-scm.com/download/win, follow installer steps.
Mac: Use Homebrew brew install git or download from https://git-scm.com/download/mac.
Linux: Use package manager, e.g. sudo apt install git for Ubuntu/Debian, or sudo yum install git for Fedora.

Configuring Git

Set your identity to record commits properly: bashCopyEditgit config --global user.name "Your Name"
git config --global user.email "your.email@example.com"

Set default editor (optional): bashCopyEditgit config --global core.editor "code --wait" # For VS Code

3. Git Basics

Core Commands

git init — Create a new Git repository in the current directory.
git status — Show status of files: untracked, modified, staged.
git add <file> — Stage changes to be included in next commit.
git commit -m "message" — Commit staged changes with a descriptive message.

Understanding the Index (Staging Area)
The staging area (index) is an intermediate area where Git collects changes before committing.
Workflow: Working Directory → Staging Area → Repository
Differences
Area
Purpose
Working Directory
Your actual files/folders you edit
Staging Area
Files prepared to be committed
Repository
The committed history (saved snapshots)

4. Git Branching and Switching

What are Branches?

Branches let you diverge from main line of development to work independently.
Helps in working on features, bug fixes, experiments without affecting main code.

Commands
git branch — List all branches.
git branch <name> — Create a new branch.
git switch <branch> or **git checkout <branch>** — Switch to a different branch.
git merge <branch> — Merge changes from specified branch into current branch.
Handling Merge Conflicts
Occurs when changes clash in the same part of a file.
Git will mark conflicts with conflict markers (<<<<<<<, =======, >>>>>>>).
Resolve manually by editing, then stage and commit.

5. Remote Repositories and GitHub

What is GitHub?

A cloud hosting platform for Git repositories.
Facilitates collaboration, code review, issue tracking.

Fork vs Clone

Clone: Copy a repository to your local machine.
Fork: Create a personal copy of someone else's repository on GitHub to contribute changes.

Commands
git clone <url> — Clone remote repo to local.
git remote add <name> <url> — Add a new remote repository.
git push -u origin main — Push commits to remote main branch and set upstream tracking.
git fetch — Download changes from remote without merging.
git pull — Download and merge remote changes to local branch.

6. Collaboration Workflow

Forking a Repo on GitHub

Fork to your account, then clone forked repo locally.
Create a new branch for your changes.

Making Changes and Pushing Branches

Work locally on your branch.
Stage and commit changes.
Push branch to your fork on GitHub.
Creating Pull Requests (PR)
On GitHub, create a PR from your branch to the original repository.
Maintainers review code, suggest changes, and merge if approved.

7. Undoing Changes

Commands
git restore <file> — Discard local changes to a file.
git reset <file> — Unstage a staged file.
git reset --soft <commit> — Move HEAD to a previous commit, keep changes staged.
git reset --hard <commit> — Move HEAD and discard changes.
git revert <commit> — Create a new commit that undoes a previous commit safely.
Recovering Deleted Files
If unstaged, use git restore <file>.
If committed and deleted, use git checkout <commit_hash> -- <file>.

8. Viewing History and Logs

Commands
git log — View commit history.
Shows commit hashes, authors, dates, and messages.
Commit Hashes
Unique SHA-1 hashes identifying commits.
Can be used to reference specific commits.
git show <commit_hash>
Displays detailed information of a commit including diffs.

9. Best Practices

Write **clear and meaningful commit messages**.
Commit **frequently** with logical chunks of work.
Keep branches **focused on a single feature or fix**.
Regularly **pull from remote** to stay updated.
Resolve conflicts carefully and test after merging.

10. Q&A and Hands-on Practice

Practice Exercises
Clone a public repository.
Make changes, stage, commit, and push.
Create new branches and merge them.
Fork a repository and submit a Pull Request.
Simulate merge conflicts and resolve them.
Use undo commands to restore and revert changes.

Ettisaf Rup,

CSE, KUET.