slide 1: why git matters

- ever lost code bc u overwrote the wrong file? **git prevents that** 😭
- wanna undo mistakes like they never happened? **git can time travel** ⏳
- need to work with others without chaos? **git keeps track of everything** ✅

---

slide 2: version control explained

✅ **tracks every change** in ur project ✅ **lets multiple ppl work on the same files** (no overwriting each other) ✅ **undo mistakes easily** (so ur flops don't ruin the project) ✅ **github is NOT git**, it's just a hosting service for git repos

---

slide 3: centralized vs distributed vcs

- centralized (CVS, SVN) → **one main server**, if it dies, u lose everything 💀
- distributed (git) → **everyone has a full copy**, so no data loss ever 😌

---

slide 4: git workflow (basic commands u need to know)

1️⃣ `git init` → start a new repo 2️⃣ `git add` → tell git to track files 3️⃣ `git commit` → save a snapshot 4️⃣ `git push`→ send changes to github 5️⃣ `git pull` → get updates from github 6️⃣ `git status` → check what's going on

---

slide 5: understanding commits

- a commit = **a checkpoint in time** 📌
- each commit has a **unique hash id**
- always write meaningful commit messages (not "fixed stuff" 💀)
- see commit history: `git log`

---

slide 6: branches explained

🌿 a branch = **a separate version of the project**

- `main` → the main stable version
- `feature-branch` → experimental work
- switch branches with `git checkout branch-name`
- create & switch with `git checkout -b new-branch`

---

slide 7: merging branches

- first, switch to `main`:

```
git checkout main
```

- then merge the branch:

```
git merge feature-branch
```

- if there are conflicts, git will ask u to fix them manually 😭

---

slide 8: dealing with merge conflicts

- git highlights conflicts like this:

```
<<<<<<< HEAD
this is ur version
=======
this is the other version
>>>>>>> feature-branch
```

- manually fix the file, then:

```
git add fixed-file
git commit -m "resolved merge conflict"
```

---

slide 9: undoing changes

- undo changes in a file:

```
git checkout -- filename
```

- undo last commit (keep changes):

```
git reset --soft HEAD~1
```

- undo last commit (erase changes forever 💀):

```
git reset --hard HEAD~1
```

---

slide 10: pushing & pulling from github

- `git push` → send commits to github
- `git pull` → get latest changes
- `git fetch` → check for updates without applying them

---

slide 11: cloning a repo

- to copy a project from github:

```
git clone https://github.com/user/repo.git
```

---

slide 12: setting up a remote repo

- if u started a local repo and now wanna put it on github:

```
git remote add origin https://github.com/user/repo.git
git push -u origin main
```

---

slide 13: tracking file history

- see detailed history of a file:

```
git log -- filename
```

- see what changed in the last commit:

```
git show HEAD
```

---

slide 14: git stash (saving work temporarily)

- save changes without committing:

```
git stash
```

- bring back stashed work:

```
git stash pop
```

---

slide 15: git rebase (cleaner history merging)

- rebasing moves ur changes **on top** of the latest version:

```
git rebase main
```

- use it to avoid messy merge commits 😌

---

slide 16: deleting branches

- delete a local branch:

```
git branch -d branch-name
```

- delete a remote branch:

```
git push origin --delete branch-name
```

---

slide 17: git revert vs reset (undoing commits properly)

- `git reset` **erases commits** from history 🚨
- `git revert` **creates a new commit** that undoes changes ✅

```
git revert HEAD
```

---

slide 18: git blame (who did what?)

- see who last changed each line in a file:

```
git blame filename
```

- useful for debugging 💀

---

slide 19: git tag (marking important commits)

- create a tag:

```
git tag v1.0
```

- push tags to github:

```
git push --tags
```

---

slide 20: git config (customizing ur setup)

- set default editor:

```
git config --global core.editor "vim"
```

- check all settings:

```
git config --list
```

---

slide 21: gitignore (ignoring unnecessary files)

- tell git to ignore files like logs & temp files:

```
node_modules/
*.log
.DS_Store
```

- put this in a `.gitignore` file

---

slide 22: git cherry-pick (picking commits from another branch)

- apply a specific commit to ur branch:

```
git cherry-pick commit-hash
```

---

slide 23: git reflog (seeing all history, even deleted commits)

- if u accidentally deleted commits, u can still find them:

```
git reflog
```

- to restore a lost commit:

```
git checkout commit-hash
```

---

slide 24: git bisect (finding the commit that broke your code)

- use binary search to find the bad commit:

```
git bisect start
```

- mark a commit as good:

```
git bisect good commit-hash
```

- mark a commit as bad:

```
git bisect bad commit-hash
```

- git helps u find exactly where the problem started 😌

---

slide 25: git worktree (working on multiple branches at once)

- make a separate folder for a branch:

```
git worktree add ../new-folder branch-name
```

- now u can work on multiple branches at the same time 🚀

---

slide 26: git hooks (automate tasks before commits or pushes)

- hooks are scripts that run before/after git actions
- example: auto-format code before committing
- stored in `.git/hooks/`

---

slide 27: git submodules (including other repos inside yours)

- add a submodule:

```
git submodule add https://github.com/user/repo.git
```

- update submodules:

```
git submodule update --init --recursive
```

---

slide 28: git clean (removing untracked files)

- see what would be deleted:

```
git clean -n
```

- actually delete them:

```
git clean -f
```

---

slide 29: git fsck (checking for repo corruption)

- check the repo for broken commits:

```
git fsck
```

- rarely needed, but useful if something goes wrong 💀

---

slide 30: git archive (exporting a repo as a zip file)

- create a zip of the latest commit:

```
git archive --format zip --output repo.zip main
```

---

slide 31: git shortlog (summary of contributions by user)

- see who committed how much:

```
git shortlog -sn
```

---

slide 32: git diff (seeing differences between commits or branches)

- show changes between commits:

```
git diff commit1 commit2
```

- show changes between branches:

```
git diff main feature-branch
```

---

slide 33: git mv (renaming files properly in git)

- rename a file and track the change:

```
git mv oldname.txt newname.txt
```

- this ensures git tracks it as a rename, not delete + add

---

slide 34: git reset vs git restore (newer way to undo changes)

- discard local changes:

```
git restore filename
```

- unstage a file:

```
git restore --staged filename
```

- `git restore` is a newer, safer alternative to `git checkout --`

---

slide 35: git show (see details of a commit)

- see what changed in a commit:

```
git show commit-hash
```

- see last commit:

```
git show HEAD
```

---

slide 36: git log with fancy formatting

- pretty commit history:

```
git log --oneline --graph --decorate --all
```

- shorter log with just commit messages:

```
git log --pretty=short
```

---

slide 37: git fetch vs git pull (the difference)

- `git fetch` downloads updates but doesn't apply them
- `git pull` downloads **and applies** updates immediately
- safer workflow:

```
git fetch
```

check changes → then do:

```
git merge origin/main
```

---

slide 38: git tag (marking specific versions of your project)

- create a lightweight tag:

```
git tag v1.0
```

- create an annotated tag (with a message):

```
git tag -a v1.0 -m "first stable version"
```

- push tags to remote:

```
git push --tags
```

---

slide 39: git revert (undoing commits safely)

- undo the last commit but **keep history intact**:

```
git revert HEAD
```

- creates a new commit that undoes the last one ✅

---

slide 40: git cherry-pick (copying a specific commit to another branch)

- apply a commit from another branch:

```
git cherry-pick commit-hash
```

- useful for grabbing individual fixes without merging everything

---

slide 41: git squash (combining multiple commits into one)

- interactively squash commits:

```
git rebase -i HEAD~3
```

- choose "squash" (s) to combine them into one clean commit

---

slide 42: git blame (who wrote this code??)

- see who last changed each line:

```
git blame filename
```

- useful for debugging & finding out who broke something 💀

---

slide 43: git fast-forward vs no-fast-forward merges

- fast-forward merges keep a linear history ✅
- no-fast-forward creates a merge commit (useful for tracking branches separately)

```
git merge --no-ff feature-branch
```

---

slide 44: git checkout vs git switch (newer way to change branches)

- `git checkout` is older and does **too many things**
- `git switch` is just for changing branches:

```
git switch branch-name
```

- `git restore` is used for undoing file changes

---

slide 45: git subtrees (alternative to submodules)

- add a repo as a subtree:

```
git subtree add --prefix=folder-name https://github.com/user/repo.git main --squash
```

- useful if u want to merge external repos **without submodule complications**

---

slide 46: git gc (garbage collection to clean up git storage)

- clean up unnecessary data:

```
git gc
```

- force aggressive cleanup:

```
git gc --aggressive
```

---

slide 47: git bundle (backup a repo into one file)

- create a bundle:

```
git bundle create repo.bundle --all
```

- restore from bundle:

```
git clone repo.bundle repo-folder
```

---

slide 48: git sparse-checkout (checkout only part of a repo)

- enable sparse checkout:

```
git sparse-checkout init
```

- specify folders to include:

```
git sparse-checkout set folder-name
```

---

slide 49: git daemon (run a git server locally)

- serve a repo over the network:

```
git daemon --reuseaddr --base-path=. --export-all --verbose
```

---

slide 50: git credentials (storing passwords securely)

- cache credentials:

```
git config --global credential.helper cache
```

- store them permanently:

```
git config --global credential.helper store
```

---

slide 51: git notes (attach notes to commits without modifying them)

- add a note to a commit:

```
git notes add -m "This commit fixes issue #42"
```

- show notes:

```
git log --show-notes
```

---

slide 52: git rerere (reuse recorded conflict resolutions)

- enable rerere:

```
git config --global rerere.enabled 1
```

- git will remember how u resolved conflicts before and **auto-resolve** next time 😌

---

slide 53: git internals (how git actually works under the hood)

- objects in git:
  - **blobs** = file contents
  - **trees** = directories
  - **commits** = snapshots
- see raw git objects:

```
git cat-file -p commit-hash
```

🚀 **DONE! u now have the full 53-slide breakdown!!** 😩💅 enjoy never having to open that dry ppt again 😭💖