✨ **git slide-by-slide breakdown** ✨

---

slide 1: motivation - why we care about git?

- u edit a file → u change it some more → u mess it up → **ur in shambles** 😭
- git lets u **rewind time** and go back to a working version 💅
- makes teamwork **less chaotic** (no more overwriting each other's code)
- flex factor: knowing git = **instant cs girlie street cred** 😌

---

slide 2: version management

✅ **version control** = keeps track of all changes in files over time (code, docs, configs, etc.) ✅ **rollback functionality** = undoes mistakes, lets u go back to a good version ✅ **branching** = lets u test new features without messing up main code ✅ **merging** = combines different changes smoothly ✅ **traceability** = logs who changed what and why (no more "who broke the code??" drama 💀)

---

slide 3: centralized vs distributed version control

🖥️ **centralized (cvs, svn)** → one server, if it dies, ur all doomed 💀 🌍 **distributed (git, darcs)** → every user has a full copy, if the main server dies, repo still survives 😌

---

slide 4: repos & working directory

📂 **repository (repo)** = storage for all versions of ur files 📝 **working directory** = where u actually edit files ✅ **commit**= saves current version to repo ✅ **checkout** = restores a previous version

---

slide 5: git architecture

🏠 **local repo** = stored on ur machine 🌍 **origin repo** = remote repo (github, gitlab, etc.) 📤 **push** = send changes from local to remote repo 📥 **fetch** = get updates from remote repo (but doesn't apply them yet) 🚚 **pull** = fetch + apply updates (most common)

---

slide 6: what is origin?

- "origin" = the default name for ur remote repo
- remote repo can be **github, gitlab, bitbucket, aws codecommit, etc.**
- lets multiple ppl work on the same code without overwriting each other

---

slide 7: staging (why git has a waiting room for files)

- `commit` applies to **all changes** in working directory
- but what if u only wanna commit some files? **staging area exists for this**
- `git add file` → moves file to staging
- `git commit` → only commits staged files

---

slide 8: creating a git repo

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

sets ur identity so commits show **ur name, not "unknown user"** 😌

```
git init
```

creates a new git repo in the current folder

`.gitignore` → a file that tells git **what to ignore** (like temp files, logs, compiled code)

---

slide 9: git status (seeing what's going on in ur repo)

```
git status
```

- shows what branch ur on
- tells u which files are modified, staged, or untracked
- files in `.gitignore` don't show up here

---

slide 10: git add (preparing files to commit)

```
git add file.txt  # add a specific file
```

```
git add .          # add all changed files
git add -A         # add all files, including deletions
```

adds files to **staging area** so they're ready for commit

---

slide 11: git commit (actually saving changes)

`git commit -m "commit message"`

saves staged changes as a new version in git **(like a checkpoint in a game)**

`git commit --amend`

edits last commit instead of making a new one (if u forgot something)

---

slide 12: git log (checking commit history)

`git log`

- shows all past commits
- commit id (hash) = unique id for each commit
- `git log --oneline` → shorter version of log
- `git log --graph --decorate --all` → pretty visual of branches

---

slide 13: git diff (seeing what changed)

`git diff`

- shows line-by-line differences between file versions
- `git diff HEAD` → compares current work to last commit
- `git diff commit1 commit2` → compares two commits

---

slide 14: git show (seeing details of a commit)

`git show <commit>`

- displays details of a specific commit (message, changes, author, date)
- `git show HEAD` → shows last commit

---

slide 15: git checkout & git restore (undoing changes)

```
git checkout commit-id  # go back in time
```

- replaces current files with an old commit (detached HEAD state)
- use `git checkout main` to return to latest version

```
git restore filename  # undo changes in a file
```

- use this instead of `checkout` in newer git versions

---

slide 16: branching (parallel universes for code)

```
git branch new-feature  # create a new branch
git switch new-feature  # move to that branch
git checkout -b new-feature  # create & switch at the same time
```

- each branch = independent version of code
- main branch stays clean while u experiment

---

slide 17: merging (bringing branches together)

```
git checkout main  # switch to main
git merge new-feature  # merge the branch into main
```

- keeps main updated with new changes
- sometimes causes merge conflicts (which u have to fix manually)

---

slide 18: syncing with remote repos (github, gitlab, etc.)

```
git remote add origin your_repo_link
git push -u origin main
```

- pushes local commits to github
- `git pull origin main` → pulls latest changes from github

---

slide 19: git reset & git revert (undoing commits)

```
git reset --hard HEAD~1  # erase last commit completely
git revert HEAD  # undo last commit, but keep a history of it
```

- `reset` = **fully delete a commit**
- `revert` = **makes a new commit that undoes the last one**

---

ok bestie this is now ur ultimate git study guide 😌💅 go be the **realest cs girlie in that lab** and flex ur git knowledge 😩💖