GIT & GITHUB

- git init : initialize git for a project

- git remote add origin <link> : set up a remote repository for the project
  - origin : an alias to a particular resp

- git revert <commit's hash> : revert a commit
  (git revert -m 1 <commit's hash> if the commit is a merge)

- git commit : save changes in the local repository

- git branch : a pointer to a snapshot of our changes
  - -f for moving a branch into another commit
- git checkout : checkout A means we move to branch A

- git merge <branch A> : merge branch A to the current branch
  - conflict : when we try to merge 2 branches with diff contents

- pull request : request to merge a branch with the main branch

- git ignore : what file (directory) in .gitignore will not be pushed to remote
  - Fix the remote resp to comply with .gitignore : git rm -r --cached <file> → push

- git cherry-pick : pick commits and add it to current HEAD (the commit just check out)

- git reset : reverse changes by moving a branch ref backward in time (only work for local)

- git revert : reverse changes for remote by creating a modified commit

(don't create new commit)

- git commit --amend : combine staged changes within the last commit

- Trunk - Based Dev : Work on main branch only, and the code base is always up-to-date and stable
  - Pros : early issue detection, quick feed back, reduce overhead
  - Cons : conflicts & integration issues, robust testing, hard to undo

- Feature Branching : Creating a new branch for a specific feature / change, then merge back to main using pull request
  - Pros : parallel dev, good management, main stability, encourage short-lived branches, assist testing process
  - Cons : big # branches, delay merging for review, conflicts due to branch dependencies

comes from stakeholders, application field, documentation

REQUIREMENTS ENGINEERING

- Non-functional req : criteria used to judge how the system perform, not what it does (ex : security, accuracy, cost, reusability, ...)

- Completeness : what the system need to do, determining the relevance of req

- User req : written for customers, no technical details (the software can do this that)

- System req : written for devs, include functional/ nonfunctional reqs
  - specific, clear, have technical details

Ex : the user should be provided with facilities to define the type of external file

- Analyzing req : verification → validation → risk analysis
  (complete, pertinent) (customer wants?) (error?)

- Req prioritization : mandatory → nice to have → useless