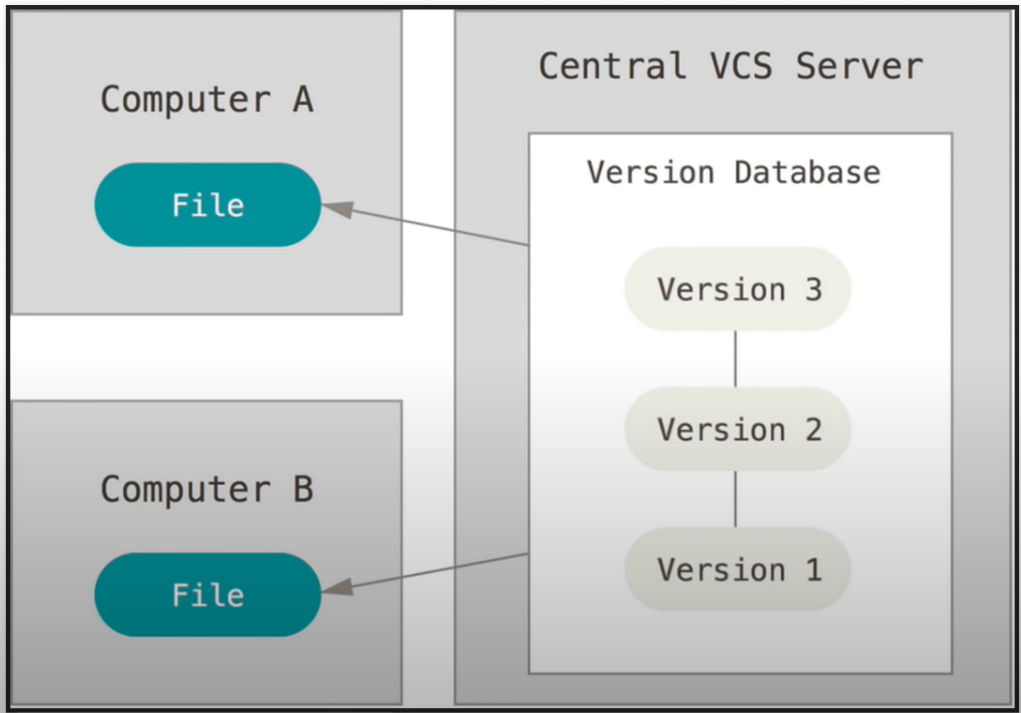
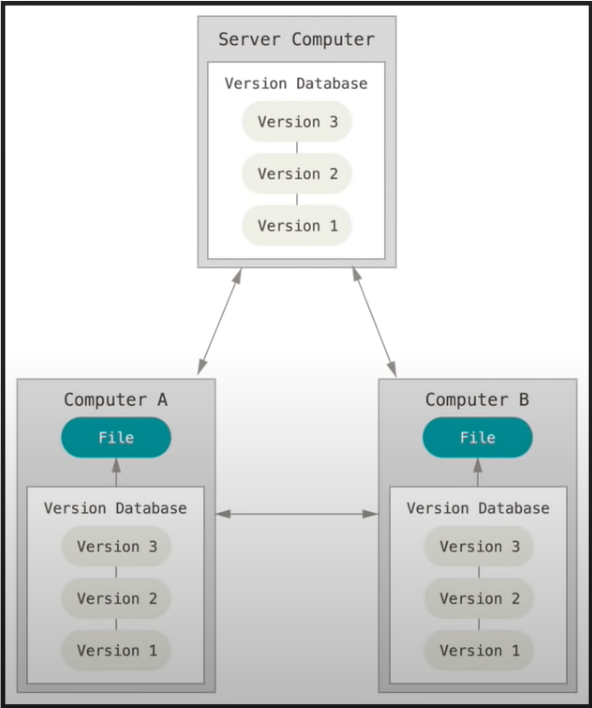
**Git**

Difference between central and distributed version control system (VCS)

* Central VCS (e.g. SVN):



* + central data is located in one place
  + people check out from a central location make changes and check back in
  + Problem:
    - if there is no access to the central server / an internet connection
    - if the central repo gets corrupted / deleted the only solution is a backup
* Distributed VCS (e.g Git):



* + everybody has a local repository which has all the information the remote repository has based on the last time one synced
  + if you have no access to the remote repository, you can still view every single change that has been made tot he repo since it has been created
  + everyone has a „backup“ oft he entire repo

First Time Setup

* git –version = shows which version is installed
* set config values:
  + git config –global user.name „username“
  + git config –global user.email „email-adress“
  + git config –list //to check
* help with commands:
  + git help <command/verb>
  + git <command/verb> --help

Getting Started

Two scenarios:

1. existing project on local machine:
   1. initialize a repo from existing code:
      * git init //from within the directory
      * git status
   2. to ignore files:
      * touch .gitignore //simple textfile to tell git which files to ignore
   3. add files tot he staging area:
      * git add -A //add all files to the staging area
      * git add <filename> //add a single file to the staging area
      * git status //to check
      * git reset <filename> //to remove a single file from the staging area
      * git reset //remove all files from the staging area
   4. committing files:
      * git add -A or git add <filename>
      * git status
      * git commit -m “Useful and detailed commit message“
      * git status
      * git log
2. existing project on remote repository
   1. cloning a remote repository:
      * git clone <url> <path where to clone>
   2. viewing information about a remote repository
      * git remote -v
      * git branch -a

Pushing Changes to a remote repository

* git diff //shows the changes that have been made
* git status //shows the state oft he files in the repo
* git add -A / git add <filename>
* git commit -m “Useful commit message“
* pushing to the repository:
  + git pull origin master //to get all the changes that might have been made to the

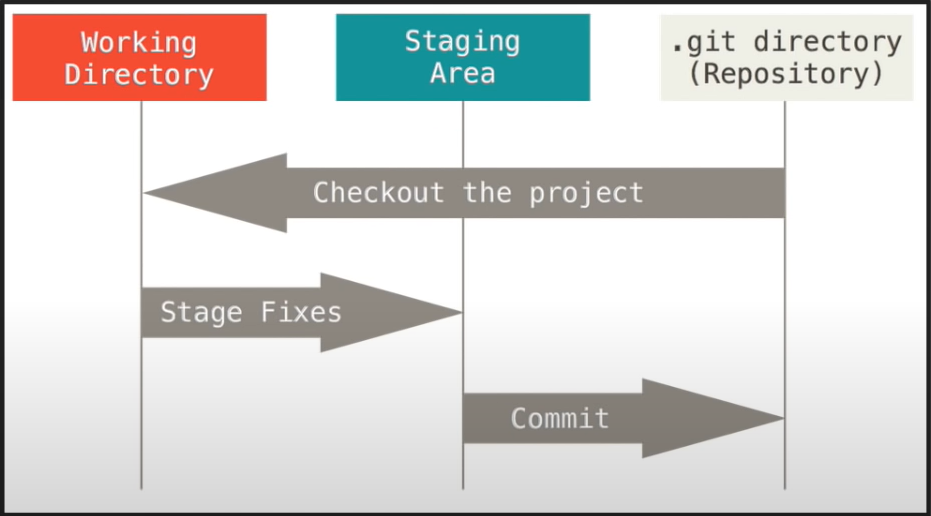
repository since the last time you pulled form the repo

* origin = name oft he repository
* master = name oft he branch
  + git push origin master

Common Workflow

* create a branch to work on your desired feature
  + git branch <name oft he branch>
  + git checkout <name oft he branch>
  + git branch //shows all available local branches
* work on your code
* git add -A
* git commit -m “message“ //commited to local branch, no effect on the remote repository!
* push the branch to the remote repo:
  + git push -u <repo-name> <branch-name> // the -u flag “registers“ the local branch with the remote branch
  + git branch -a //show all branches (local and remote)
* merge a branch:
  + git checkout master //select the master branch
  + git pull origin master //pull all changes made to the master branch
  + git branch –merged
  + git merge <branch you want to merge>
  + git push origin master
* deleting a branch:
  + git branch –merged //verify the branch u want to delete is merged
  + git branch -d <name of the branch to delete>
  + git branch -a //you can see the remote branch is still there
  + git push origin –delete <name of the branch to delete> //delete remote branch

Important states in Git-workflow



* working directory:
  + untracked or modified files //will be listed with “git status“
* staging area:
  + organize what one wants to be commited
  + pick and choose what to commit
  + if you have worked on a lot of files stage the ones that belong together and commit them in logical chunks tob e more detailed in the commits
* commited files

**FAQ:**

* To undo any uncommited changes to a file, just checkout that file:
  + git checkout <filename>
* Correct a misleading commit message:
  + git commit –amend -m “correct commit message“

//changes the git history only use this if you are the only one that had access to the changes i.e. if you didnt push

* forgot a file in your commit:
  + git add <filename oft he file you forgot>
  + git commit –amend

//changes the git history aswell!

* comitted to the wrong branch:
  + git log and copy the hash oft he commit (first 6 characters should be enough)
  + git checkout <branch you wanted to commit to>
  + git cherry-pick <hash you copied>
  + git log
  + git checkout <branch you accidentially commited to>
  + git log //grab the hash oft he commit before your false commit
  + git reset [soft|mixed|hard]
    - git reset –soft <copied hash> //keeps changes made in the staging directory
    - git reset –mixed <copied hash> // default option – kept the changes but in the working directory
    - git reset –hard <copied hash> //reverts all changes to the state oft he commit of which we copied the hash (doesn’t get rid of untracked files)
      * reverse a hard reset:
        + git **reflog**
        + grab hash before the reset
        + git checkout <copied hash>
        + git branch backup //to save the reverted changes, because we induced a detached head state
  + get rid of untracked files:
    - git clean -df // -d deletes untracked directories, -f untracked files
* undo changes others have already pulled:
  + git revert //creates new commits, that undo changes
  + git log //copy hash of commit you want to undo
  + git revert <hash>

Notes:

detached head state – will get garbage collected (deleted)

git diff <commit-hash1> <commit-hash2> - shows all the differences between two commits