CSE3CWA/CSE5006
Cloud-based Web Application

Dr. Kiki Adhinugraha

Lecturer and Schedule

- Lecturer
  - Dr Kiki Adhinugraha
  - K.Adhinugraha@latrobe.edu.au
  - PS1-215A
- Schedule
  - **Lecture** : Tue, 13-15, ED2-01-104 (CSE3CWA & CSE5006)
  - **Labs** CSE3CWA:
    - Wed 09-11 BG-01-103
    - Wed 11-13 BG-01-103
  - **Labs** CSE5006:
    - Wed 09-11 BG-01-104

  Notes: No Labs in Week 1
  - **Consultation** : TBA

Topics

- Part 1: Cloud Repository and Virtualisation (Week 1,2)
  - Cloud-based Version Control
  - Virtual Machine
  - Containers
- Part 2: Web Application Development (Week 3 – 7)
  - Website and Javascript
  - Interactive frontend using React Library
  - ORM
  - API

Topics

- Part 3: AWS Platform (Week 8-11)
  - AWS Fundamentals
  - AWS Cloud Infrastructure
  - AWS Containers
  - CI/CD in AWS

Individual Assessments

Assessment #	%	Type	Due
1	25%	Quiz in LMS (Closed Book)	Week 3 (Part A – 10%) Week 11 (Part B – 15%)
2	25%	Build a web-based application	Week 11
3	50%	Paper-based exam (Closed book)	Exam week

Check LMS for the update

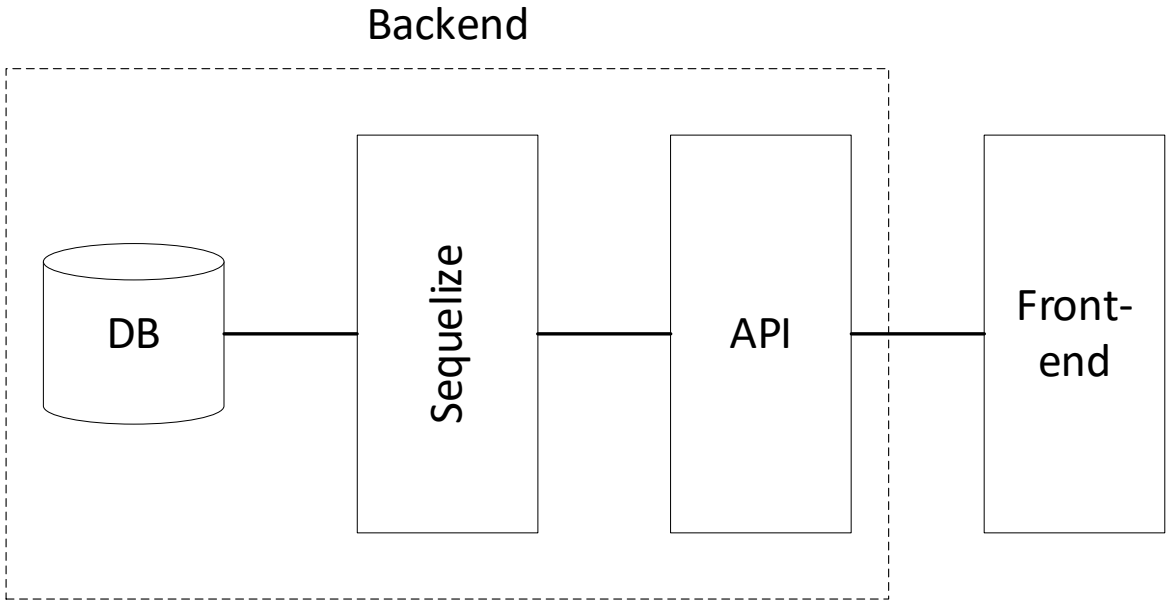
Assessments

- Assessment 1 (20%)
  - Assessment 1A (10%) – LMS Quiz, Week 3 (45 mins)
  - Assessment 1B (15%) – LMS Quiz, Week 11 (60 mins)

  - Individual - Closed Book
  - MCQ
  - Randomised questions
  - 1 attempt only
  - Safe Exam Browser must be activated

Assessments

- Assessment 2 (25%), Week 12
  - Develop a fully working cloud-ready web application that has database and ORM, API, and an interactive frontend that is deployed using a container with microservices.



Assessment 3

- Paper-based examination
  - Exam week
  - Centrally managed by university
  - Closed-book & Individual
  - Contains:
    - 15 MCQ
    - 5 Essays

Academic Integrity

- https://latrobe.libguides.com/academic-integrity/what-is-academic-integrity
- Academic integrity means putting those values into practice by:
  - being honest in the academic work you do at university
  - being fair to others
  - taking responsibility for learning, and following the conventions of scholarship.
- It is the University's responsibility to award credit for honestly conducted work, and it is your responsibility to ensure that you demonstrate academic integrity.

Academic Misconduct

- Types of academic misconduct include
  - plagiarism (copying other people's work without proper acknowledgement)
  - collusion (working together on an individual assessment task)
  - Cheating
- All academic misconduct allegation will be processed by Academic Integrity Unit. All marks will be withheld until further notice.
- There will be penalties for academic misconduct. https://www.latrobe.edu.au/students/admin/academic-integrity/penalties-for-academic-misconduct

Student Feedback on Subjects (SFS)

- It is part of the quality assurance process that occurs across the university.
- Students are invited to tell us about your learning experiences in this subject.
- Please tell us of your experience in this subject.
- Your views will be taken seriously and will assist us to enhance this subject for the next group of students.
- Your feedback will also contribute to the text for "Summary of Previous Student Feedback" below so please take the time to tell us your views.
- The surveys are anonymous and will be distributed prior to the end of the teaching period.

How to pass this subject

- Attend lectures
- Complete lab classes
- Submit assignment (Do not COPY)
- Attend exam
- Practice… Practice… Practice…

SILO

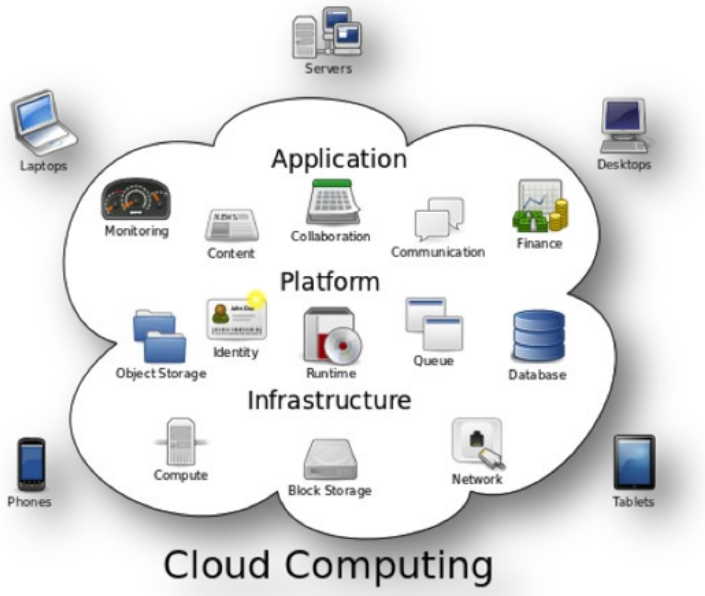
- Upon successful completion of this subject, you should be able to:
  1. Design and Develop web applications using Javascript
  2. Design and build a stateless web server based on cloud technologies
  3. Design and customize backend Web applications based on user requirements
  4. Use modern software engineering tools to build and deploy robust code for scalable web sites.
  5. Investigate storage technologies to determine optimal choice for a web site.

Code repository systems and version control using Git

Week 1

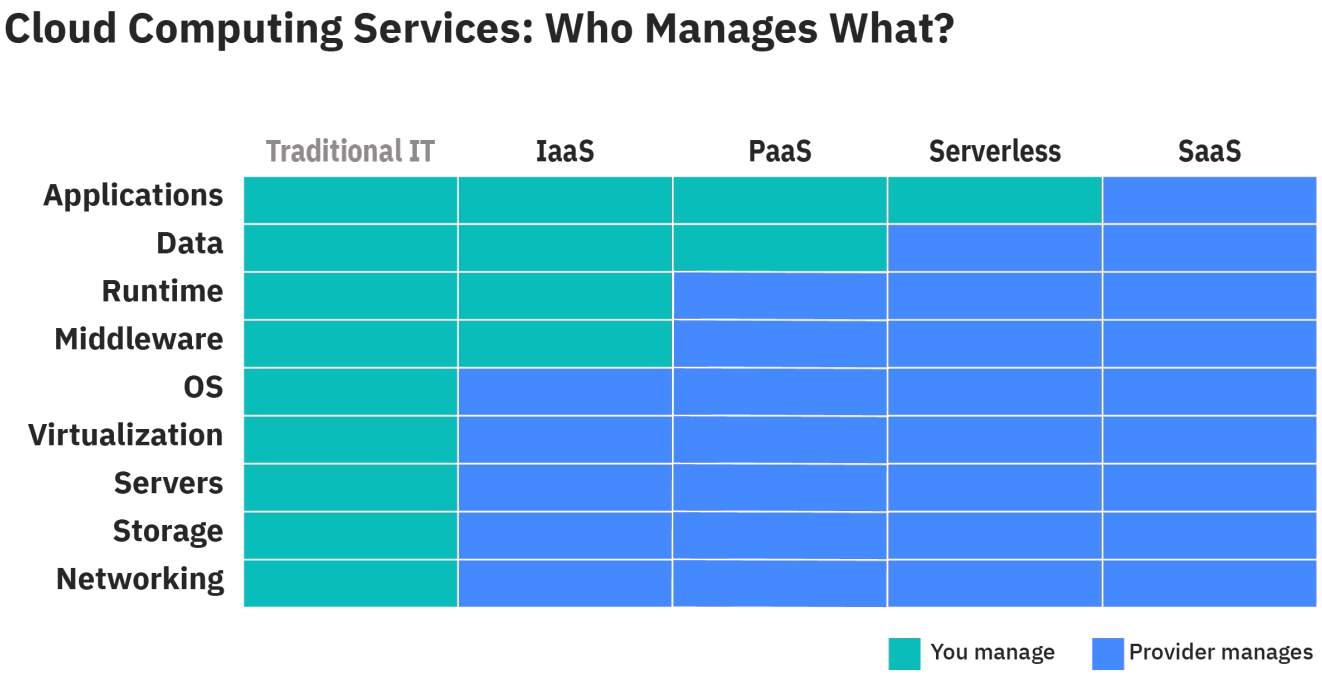
What is Cloud Computing?

- The cloud' → servers that are accessed over the Internet
- Cloud servers are located in data centers all over the world.
- Users/companies do not have to manage physical servers



Why do we need cloud computing?

- Removes some IT costs and overhead
- Easier for companies to operate internationally
- Reliability
- More contents will be explained in Lecture 10

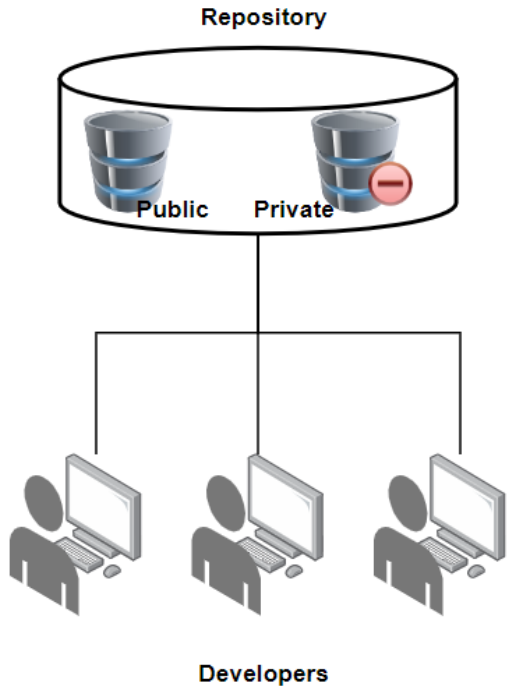


https://www.ibm.com/au-en/cloud/learn/cloud-computing

Code Repository and Version Control

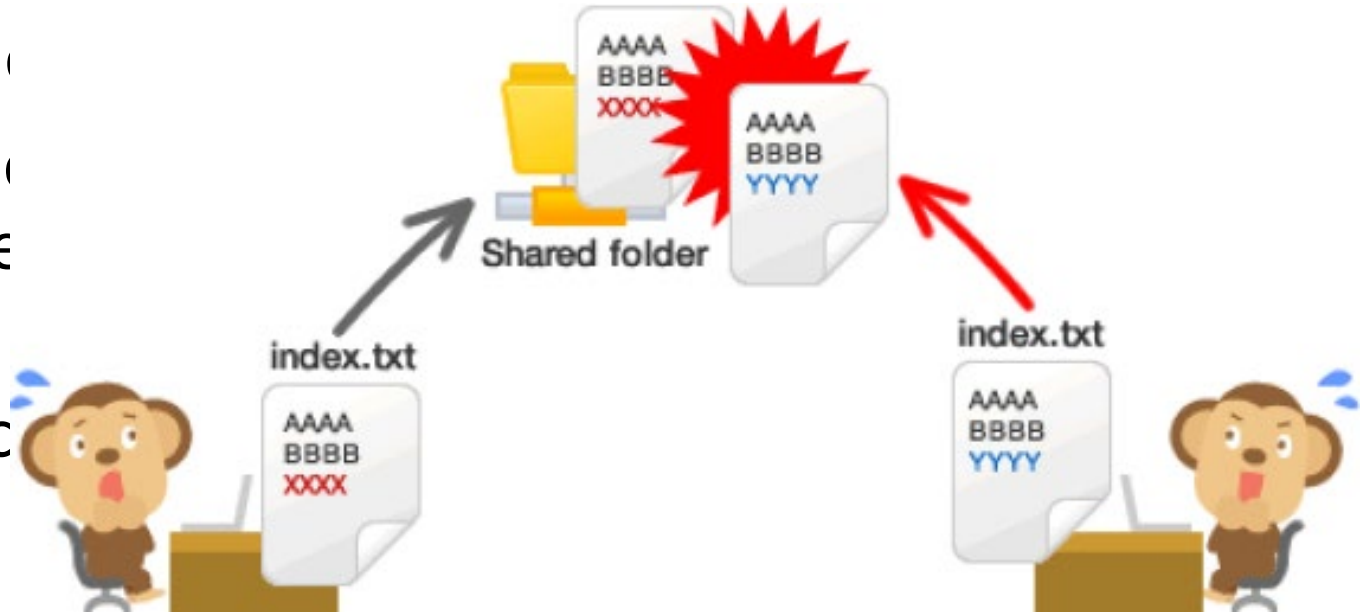
Code Repository

- Centralised storage for the source code and other related materials
- Main aim is for safekeeping
- Stored for public/private access



Version Control: Why Use it?

- Collaboration
- Imagine a team of people are working on the same set of fil
- If two people change the sam file at the same time, then the will be inconsistencies.
- People will be overwriting eac others changes.



Why use a VCS?

- A “time machine” → going back to an old version
- There is only one active project/version. Everything else (all previous versions) are neatly packed up inside the VCS.
- Understanding what happened
  - Description for every changes
- Backup
  - In the case of distributed VCS such as Git, every team member has a full-blown version of the project on his disk including the project’s complete history.
  - So even if your central server breaks down you can still get a copy from your teammates local Gitrepository.

Benefit of VCS

- A complete long-term change history of every file
  - This means every change made by many individuals over the years. Changes include the creation and deletion of files as well as edits to their contents.
- Branching and merging
  - Creating a 'branch' in VCS tools keeps multiple streams of work independent from each other while also providing the facility to merge that work back together, enabling developers to verify that the changes on each branch do not conflict.
- Traceability
  - Being able to trace each change made to the software and connect it to project management and bug tracking software such as Jira, and being able to annotate each change with a message describing the purpose and intent of the change

Types of VCN

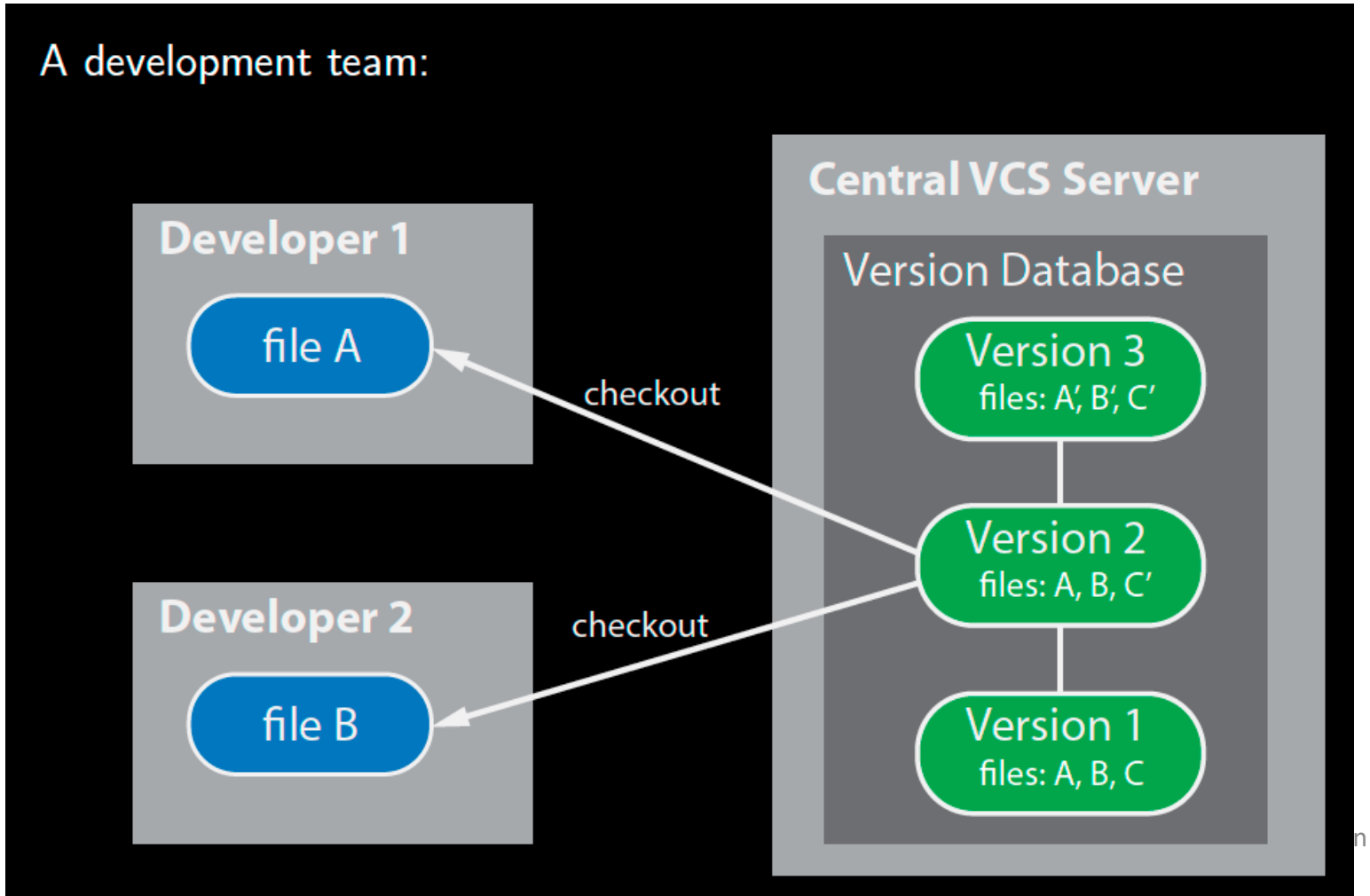
Centralised Version Control

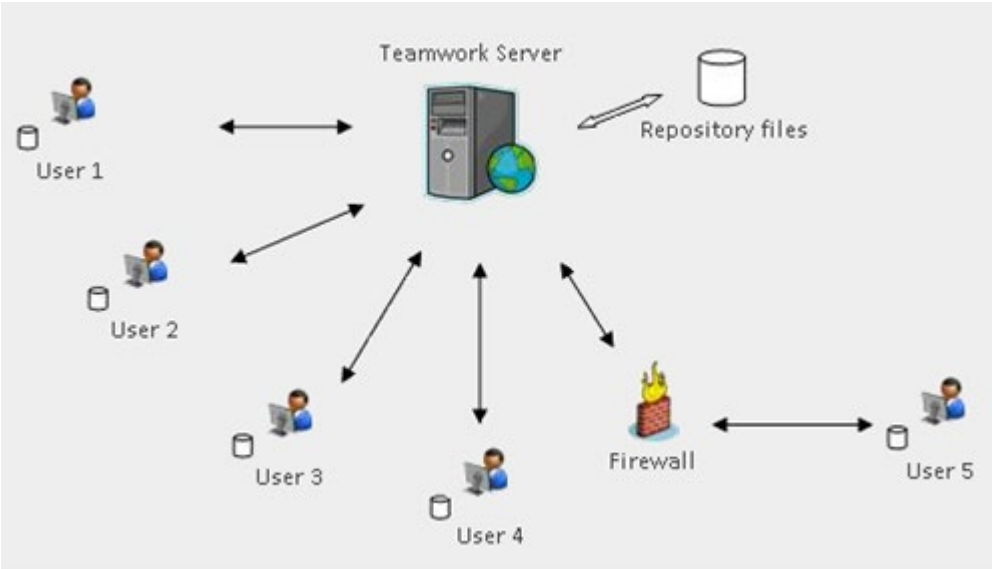
- Single shared repository
- Team members must checkout individual file
- Example:
  - CVS, Subversion (SVN), Perforce

Distributed Version Control

- No centralised repository
- Team members check out the entire repository
- Example:
  - Git, Mercurial

Centralised Version Control

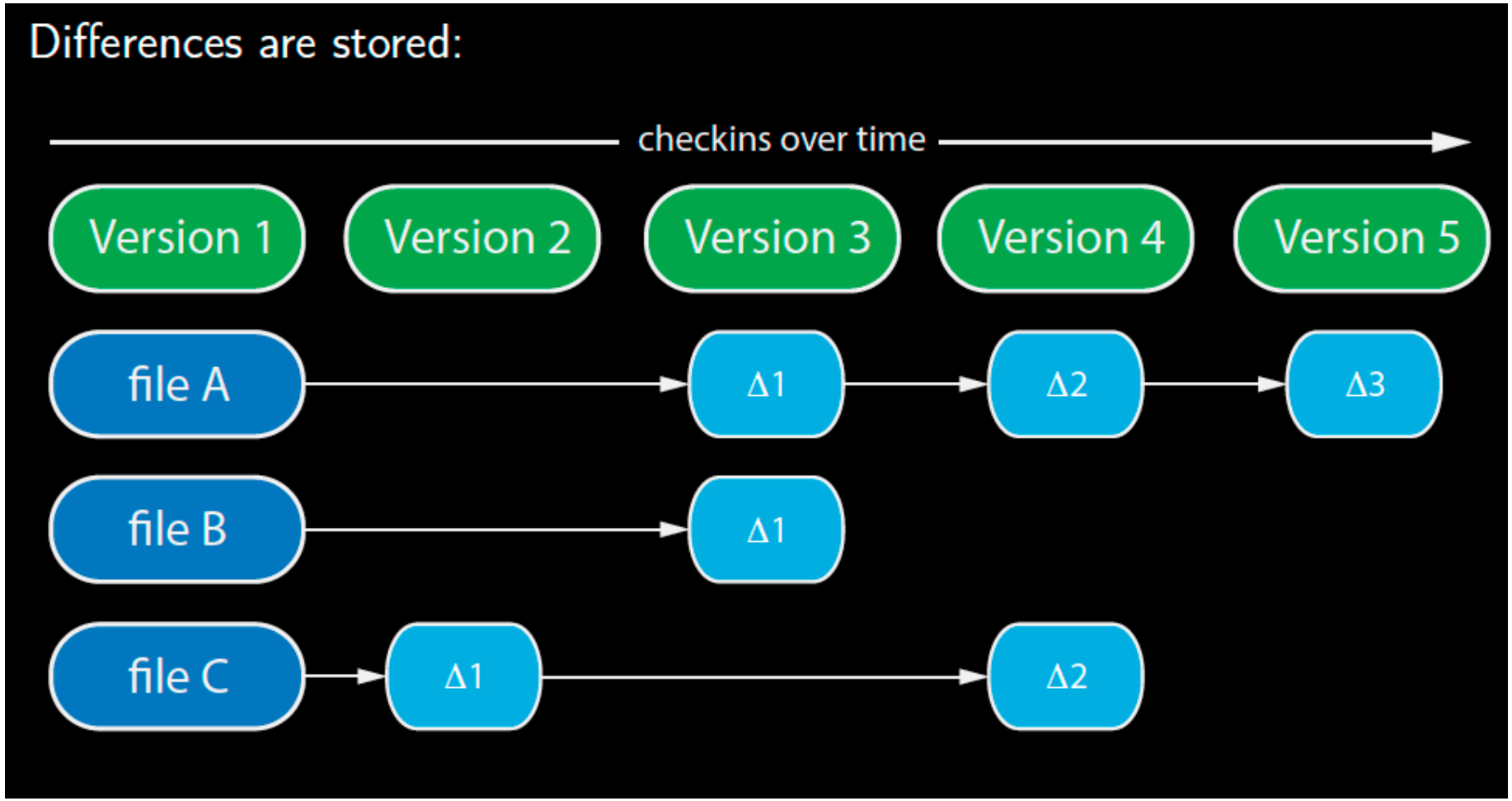




Centralised Version Control

- For a centralized version control system there is one single repository that all developers share.
- When a developer wants to work on a file he/she checks it out from the central repository.
- What happens when two developers work on the same file? Two ways to handle this:
  - **Locking** - The first person to check out the file has a lock on the file. That way prevents a later person from modifying the file at the same time
  - **Merging** – The second person who checks in the file needs to merge his/her changes with the changes checked in by the previous person.

Centralised Version Control



Problems with Centralised Version Control

- The entire project is stored only at the central server.
  - If the central server dies you lose everything
- You can not work when there is no connection to the central server
  - You cannot create different versions of your files without contacting the server.

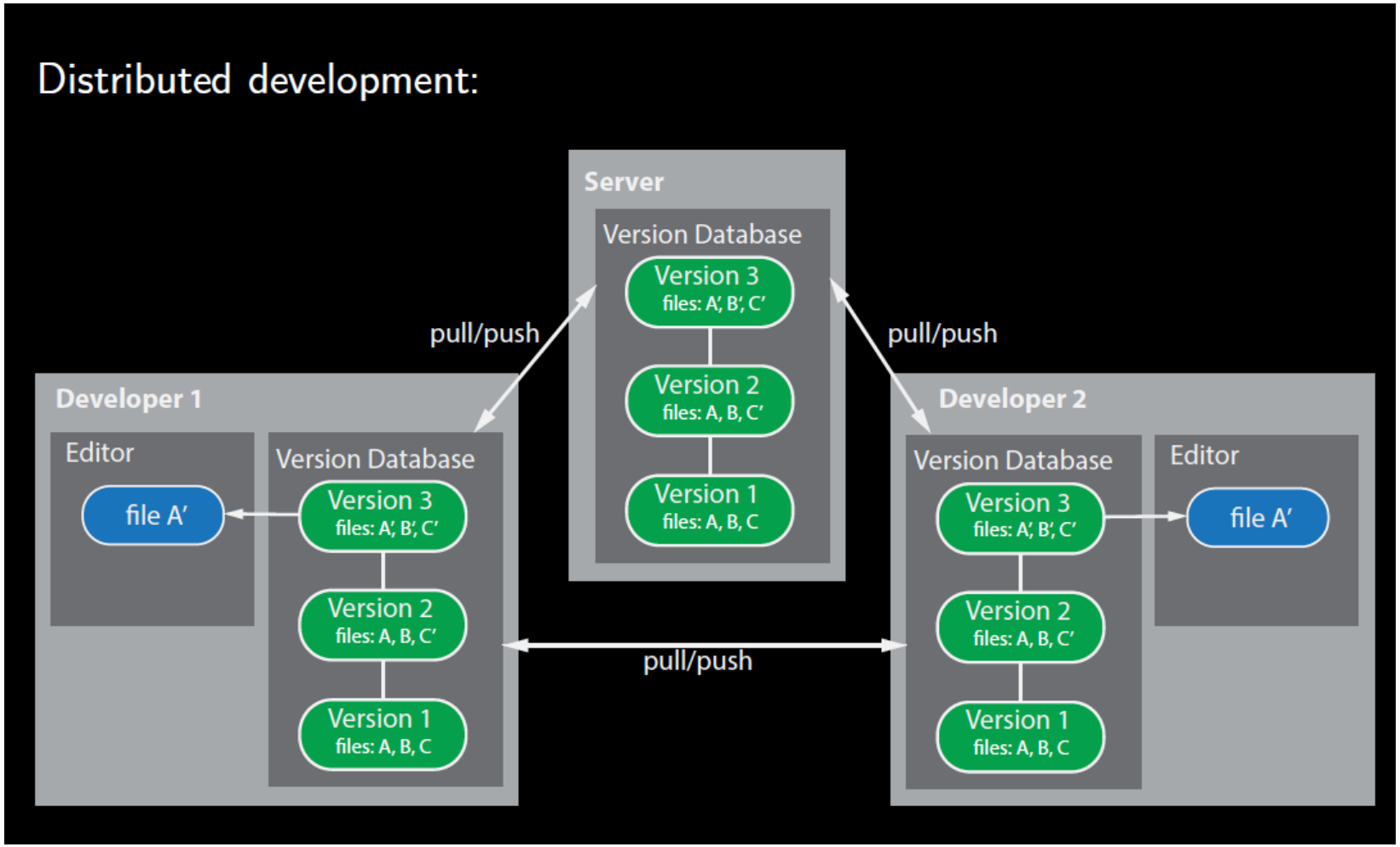
Distributed Version Control

- Distributed Teams
  - As software development has become more team-based, and teams have become more distributed, the need for distributed version control systems has emerged.
- No centralized repository
  - In distributed version control, there is no centralized repository, team members don’tcheck out individual files from these systems, they check out the entire repository.

Distributed Version Control

- The repository is not locked
  - other team members can also download the current version of a project.
- Different snapshots over time
  - Later, when a team member check the project back in, it is marked as a different version in the repository. Thus, you can think of distributed version control systems as storing different snapshots of a project over time.
- Owner determines what and how to merge
  - It is up to the owner of the repository to merge different version together if he/she so chooses.

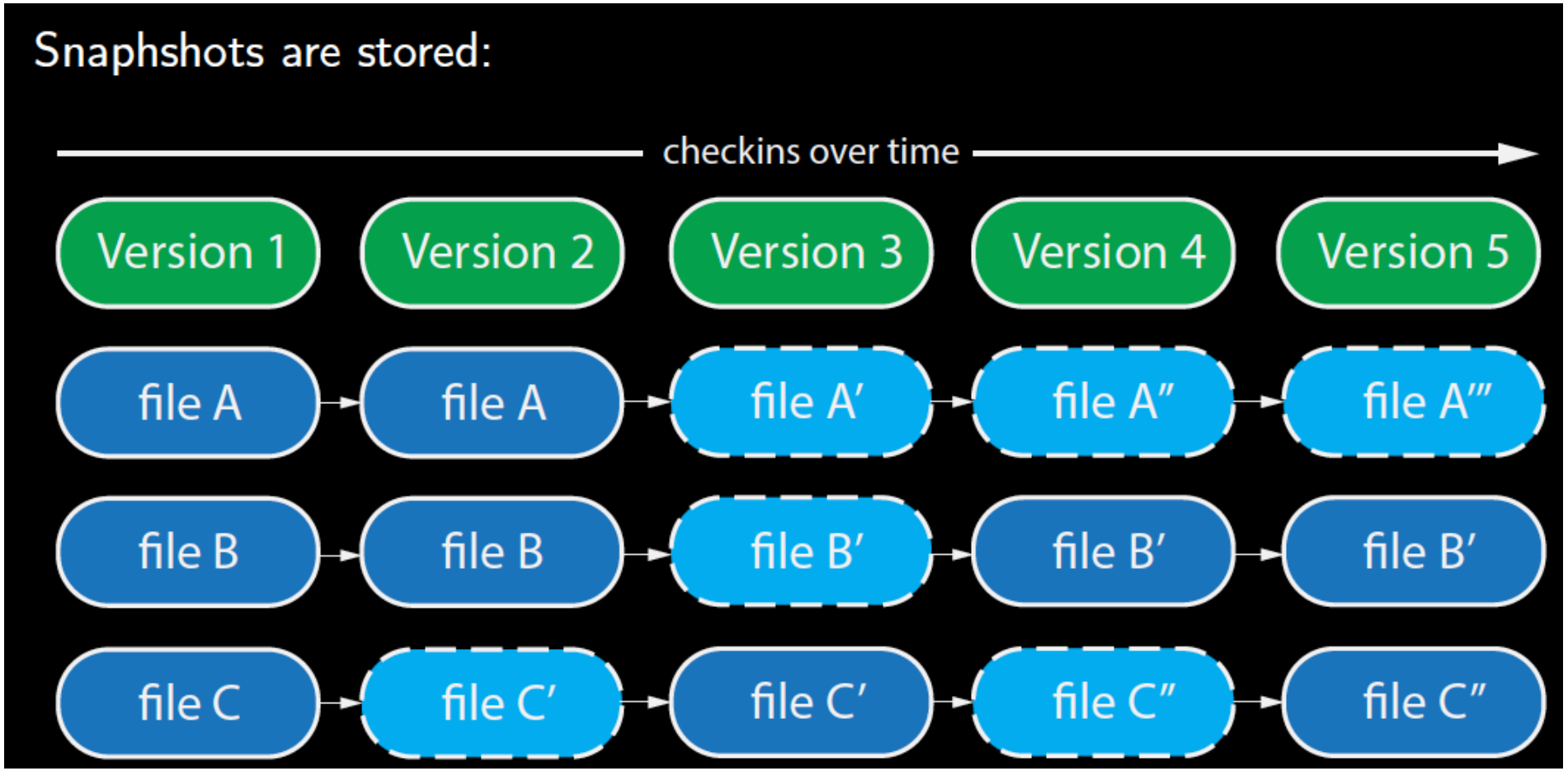
Distributed Version Control



Distributed Version Control

- Although each developer has his/her own copy of the repository. There is the idea of one person who is the owner of the repository.
- The owner is responsible for merging all the changes from all the different developers.
- A developer can push his/her changes to another developers repository.
- A developer can also pull in changes from another developers repository.

Distributed Version Control



Implementation of Version Control Systems

- Centralised
  - Apache Subversion
  - The most popular centralized version control system(CVCS) on the market
- Distributed
  - Git
  - The most widely used modern distributed version control system in the world

Apache Subversion (SVN)

- Work is comprised of three parts:
  - Trunk:
    The trunk is the hub of your current, stable code and product. It only includes tested, unbroken code. This acts as a base where all changes are made from.
  - Branches:
    Here is where you house new code and features. Using a copy of the trunk code, team members conduct research and development in the branch. Doing so allows each team member to work on the enhanced features without disrupting each other's progress.
  - Tags:
    Consider tags a duplicate of a branch at a given point in time. Tags are not used during development, but rather during deployment after the branch's code is finished. Marking your code with tags makes it easy to review and, if necessary, revert your code.

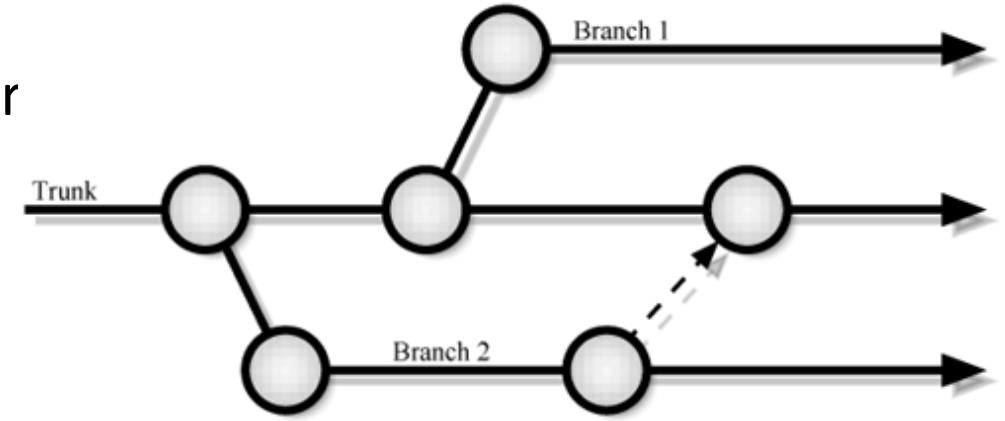
Apache Subversion (SVN)

- Every commit pushed to the server requires a new version of the entire repository, including the unchanged files. Pushing a commit may require both the server and client to update before the commit can go through. This is necessary in cases where a locally changed file has a newer revision on the server.

- The above scenario is very likely when two or more developers are working on the same file and are committing changes to it frequently.

Apache Subversion (SVN)

- SVN adopts the concept of 'branches' to isolate any code experiments or incomplete features and uses tags for code snapshotting.
- SVN also enables you to quickly retrieve versions of a code repositor through the checkout process.
- SVN also doesn’t support nested repositories.
- Subversion fails to provide the capability to visualize a complex branching tree like this.



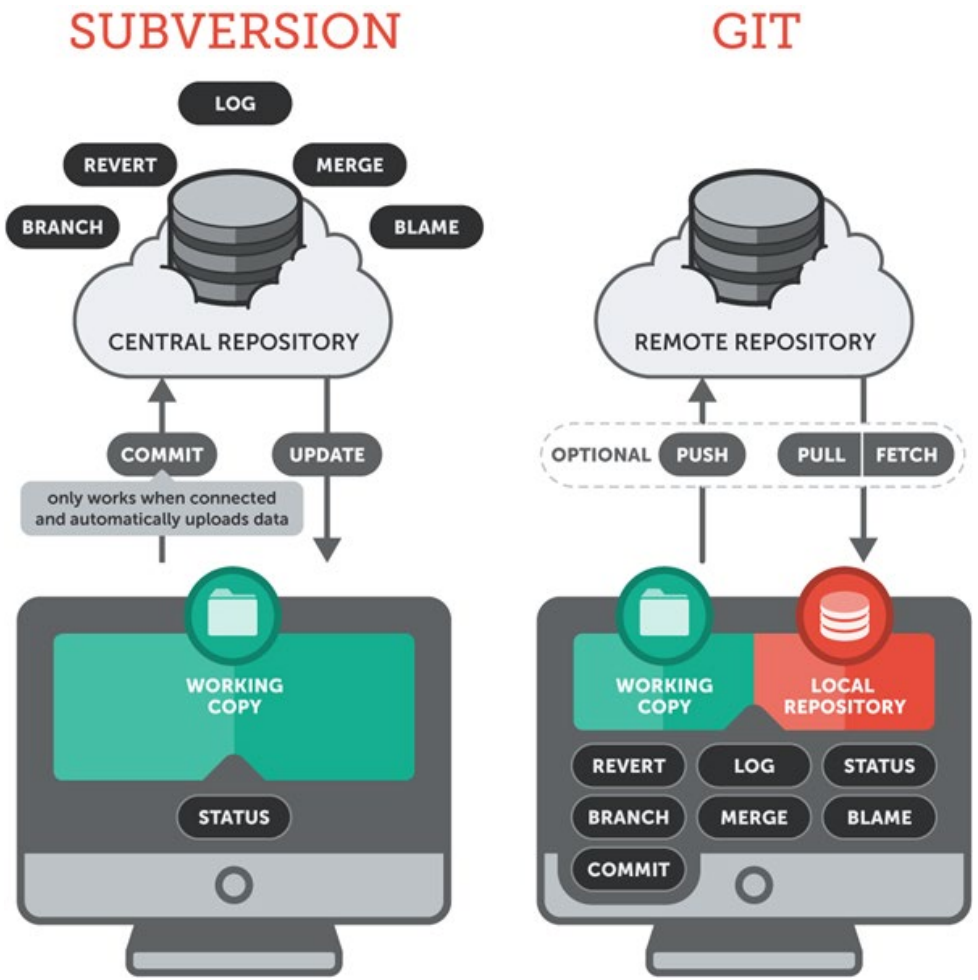
Git

- Each developer will be working with their local version of the repository instead of the central repo.
- Does not reply on active connection
- Commits are made to the local repositories before the changes are pushed to the central server.

Git – Conflict Resolution

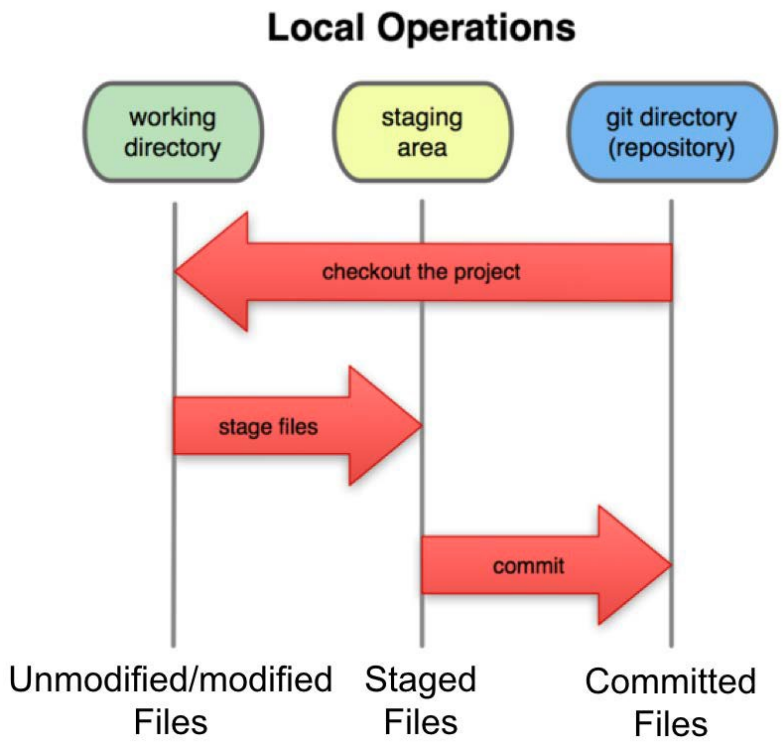
- In the scenario **when two developers want to push** their changes on the same file to the central repo, git will compare the current snapshot with the committed snapshot and allow the developer with the most updated snapshot to push changes to the repo. The changes made by the other developer are not lost. Git will instruct the developer whose push was rejected to first pull the latest changes from the central repository before pushing his/her/they changes. So, at any given point, the central repository will always hold the most up to date code.

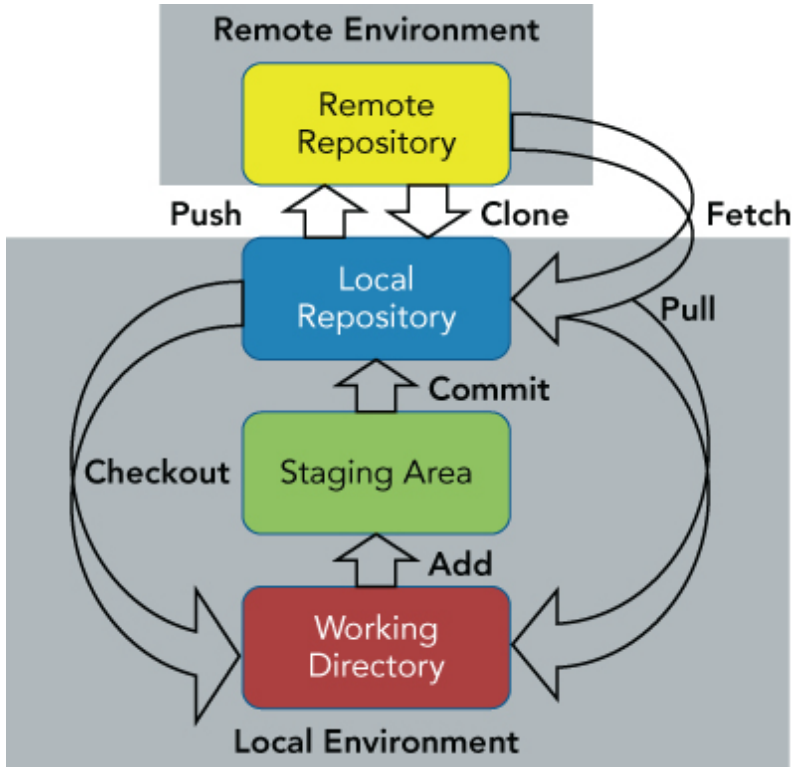
VCN Comparison



https://www.git-tower.com/learn/git/ebook/en/desktop-gui/appendix/from-subversion-to-git/

Basic Workflow

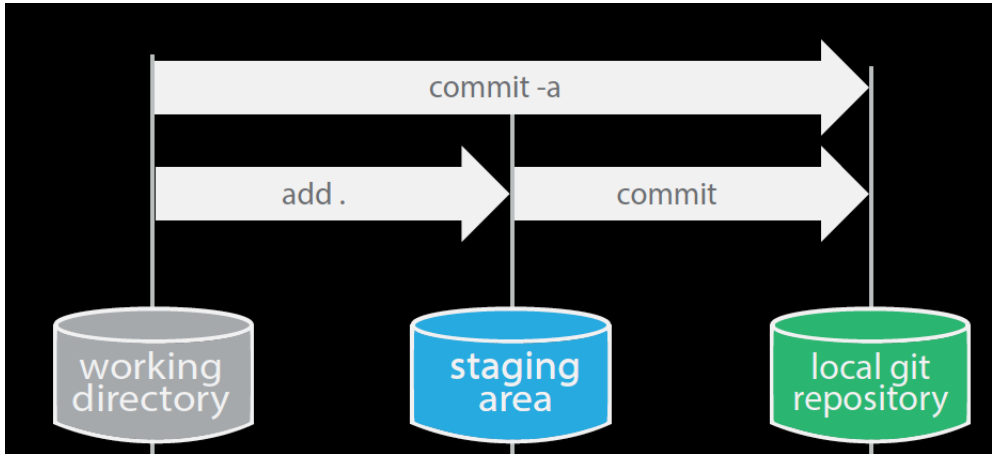




https://www.atlassian.com/git/tutorials/using-branches/git-checkout

https://kelvinleong.github.io/git/2017/01/20/Use-Git.html

Committing all files in a directory at once



- There are two ways of committing the entire directory
  - Using staging area
    - gitadd .
    - gitcommit -m “made big change”
- Directly commit into repository bypassing the staging area
  - gitcommit -a -m “made big change”

Setting up Git

- Follow the tutorial in LMS
- Set the name and email for Gitto use when you commit:
  - `$ gitconfig--global user.name“Bugs Bunny”`
  - `$ gitconfig--global user.emailstudent@gmail.com`
- You can call git config–list to verify these are set.
  - These will be set globally for all Git projects you work with.
- You can also set variables on a project-only basis by not using the `--global` flag.

Creating and Using a Repository

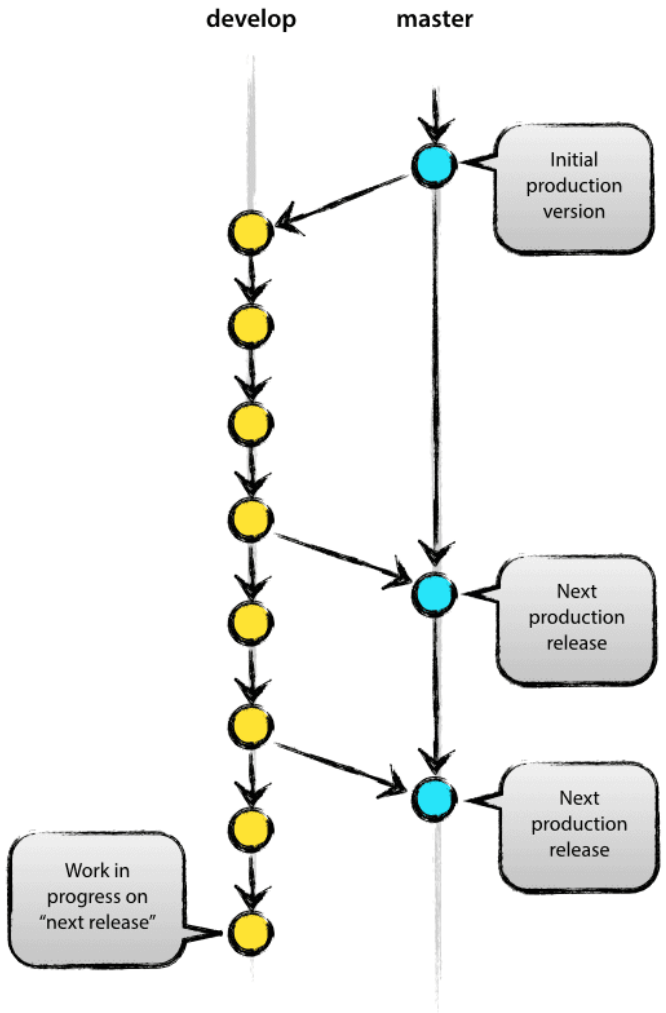
- To create a Gitrepo in your current directory:
  - $ git init
    - This will create a .git directory in your current directory.
- Then you can commit files in that directory into the repo by first adding to the staging area and then committing into the repo:
  - $ git add file1.java
  - $ git commit –m “initial project version”
- You can use the following to unstage a file
  - $ git reset HEAD file1.java
- You can get back the original version of a file that you have modified using the following
  - $ git checkout file1.java

Creating New Branches

- Git allows you to diverge from the main line of development.
- Allows you to work without messing up the main line of code.
- Better than creating a new copy of your source code directory for each new branch.
- Main development branch = master.
- New versions may be created along the master branch, or new branches can be created off of it.
- The changes to the new branches will be tracked independently.
- The branches can possibly be merged back into the master branch.

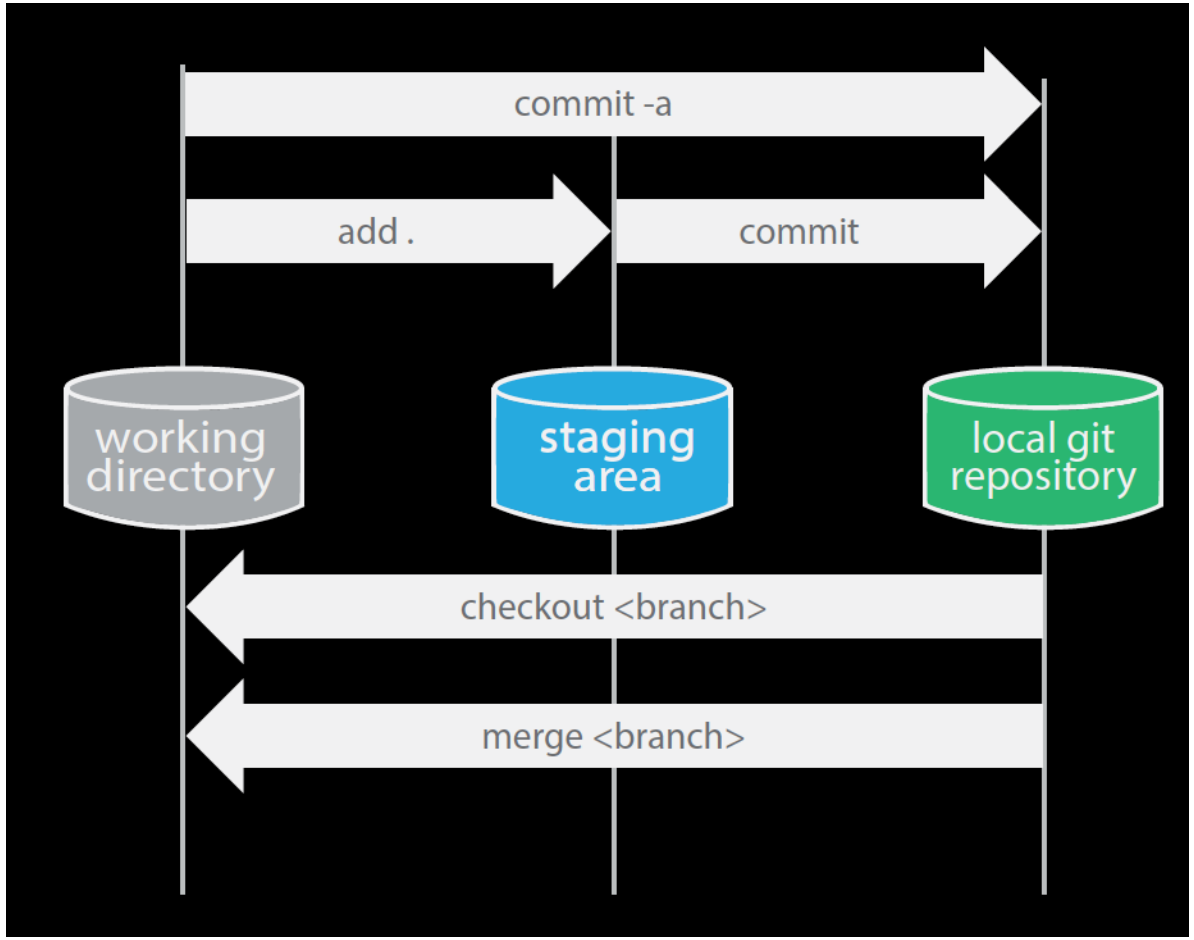
Git Branching

- In this example there is a master branch which only contains production versions of the code
  - Production code is code that is actually used to serve customers. It is usually heavily tested code.
- The other branch is the develop branch that is used by the developer to work on development version of the code
- When the code is of production quality it is merged into the master branch.
- The owner controls what goes into the master branch.



Checkout and merging a branch

- To create a branch called develop
  - $ git branch develop
- To list all branches (* shows which one you are currently on)
  - $ git branch
- To switch to the develop branch
  - $ git checkout develop
- To switch back to the master branch and then merge the changes from the develop branch do the following
  - $ git checkout master
  - $ git merge develop

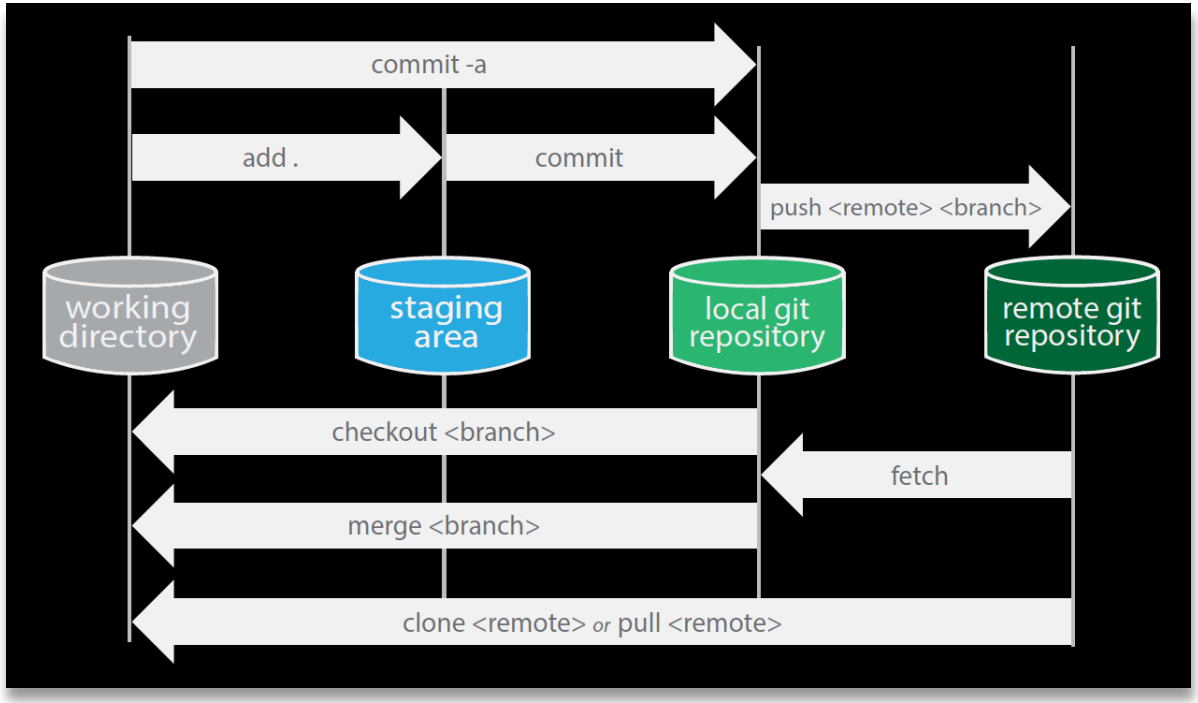


Forking

- You may really like an open source Gitproject and want to make a lot of changes to it.
- Also your changes may not be wanted by other people. Therefore the owner of the original Git repository may not want to merge your changes to the open source project.
- In this case you can fork the project.
- This will clone the entire project into a new project.
- You can then have full control of the forked project.
- From here on if you want to push changes back to the original repository that will be hard.

Remote Git Repository

- In order to have multiple developers working on the same repository we need to create a remote git repository.
- Developers push their change to the remote repository
- We can clone or pull the data from a remote repository into our working directory.



Create and Push to Remote Repository

- Lets assume you have already created a local git repository.
- Now here is how you can add the remote repository
  - $ git remote add origin <Remote URL>
    - Here originis the name you give to the remote repository. After this command every where you use origin it will refer to the remote repository located at the URL specified by <Remote URL>.
    - <Remote URL> is the URL where the remote repository is located.
- Once you add the remote repository you can now push your local repository into the remote repository using the following
  - $ git push –u origin master
    - This pushes the master branch to the remote repository named origin.

Getting files from Remote Repository (fetch, pull and clone)

- The fetch command can be used to grab all the data from a remote repository
  - $ git fetch <remote-name>
- To use the above command you need to first add the remote repository into your local repository.
- If you want gitto create a local repository and also grab all the data from the remote repository in one command you can use cloneinstead
  - $ git clone <remote URL>
    - This will clone the repository stored at <remote URL>
- If you already have a copy of the remote repository but just want to update it with the latest version you can use pull
  - $ git pull <remote-name>
    - This will update the current local gitwith the remote repository specified by <remote-name>

Git commands

command	description
`git clone` ***`url [dir]`***	copy a git repository so you can add to it
`git add` ***`files`***	adds file contents to the staging area
`git commit`	records a snapshot of the staging area
`git status`	view the status of your files in the working directory and staging area
`git diff`	shows diff of what is staged and what is modified but unstaged
`git help` ***`[command]`***	get help info about a particular command
`git pull`	fetch from a remote repo and try to merge into the current branch
`git push`	push your new branches and data to a remote repository
others: `init, reset, branch, checkout, merge, log, tag`	

Thank You