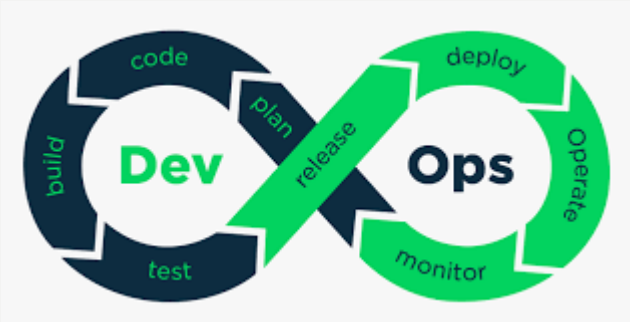
DEVOPS FUNDAMENTALS



DAY- I

DevOps: Background

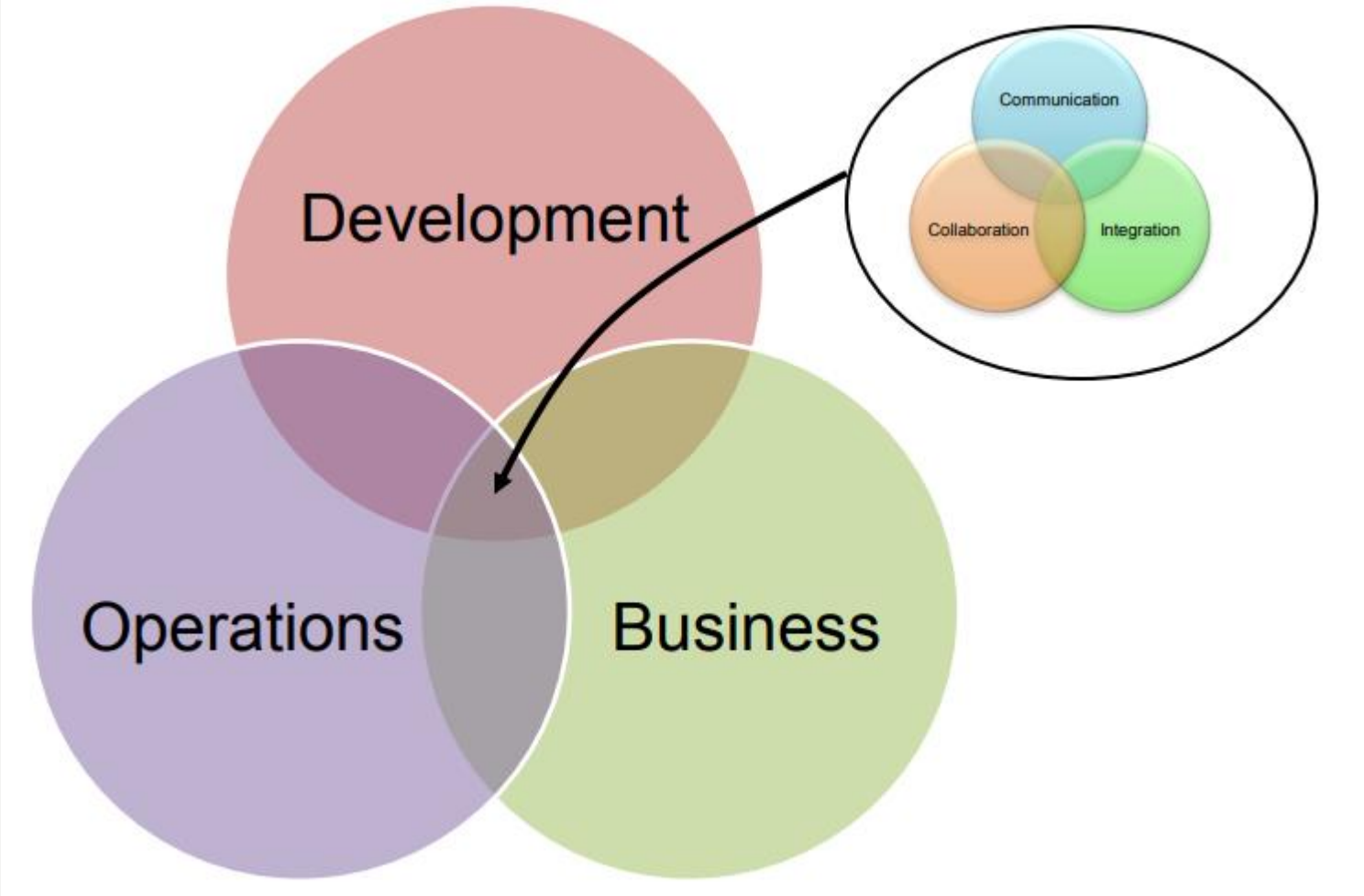
DevOps

NOTE:

a

DevOps: Background

DevOps

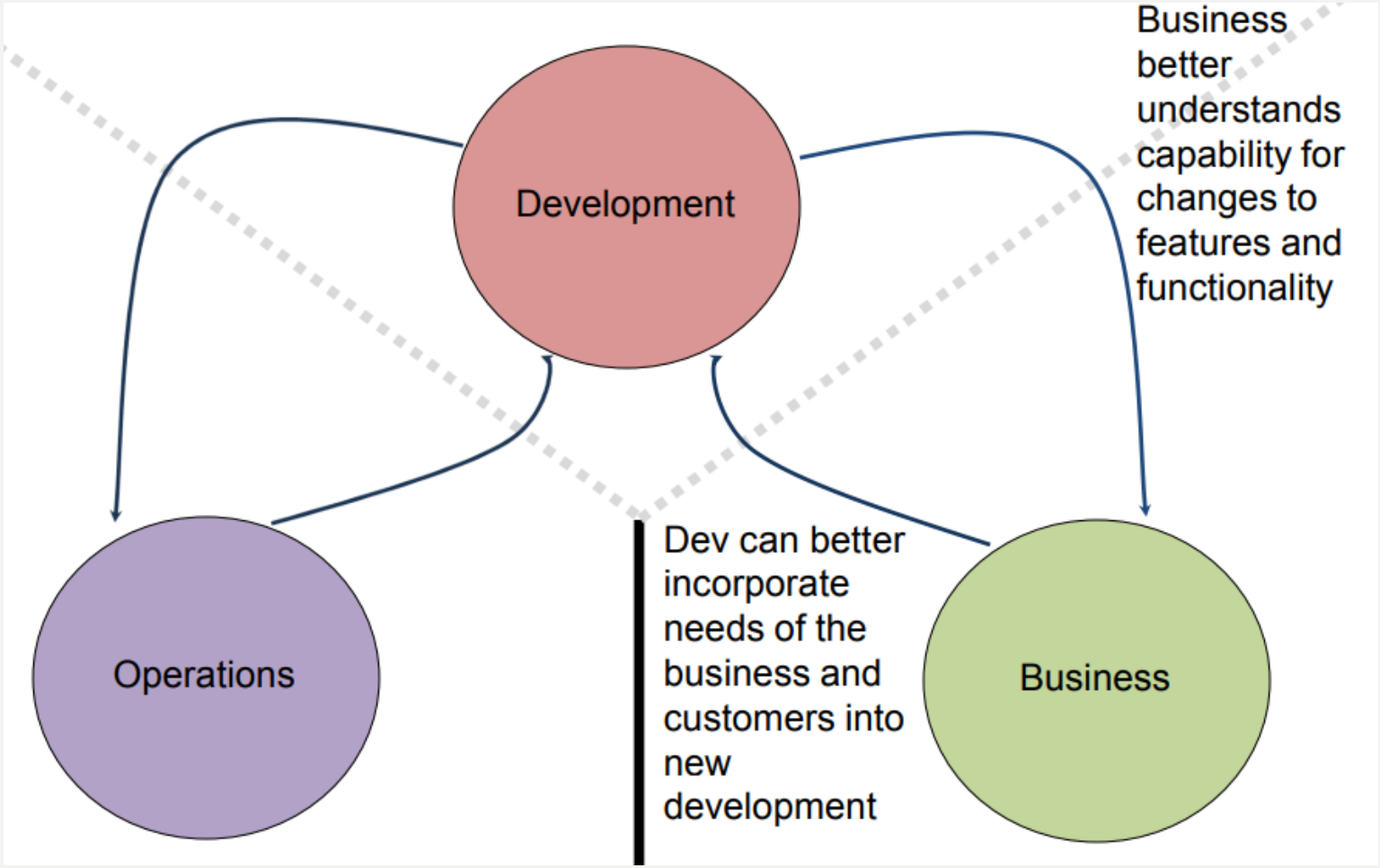


NOTE:

a

DevOps: Background

DevOps

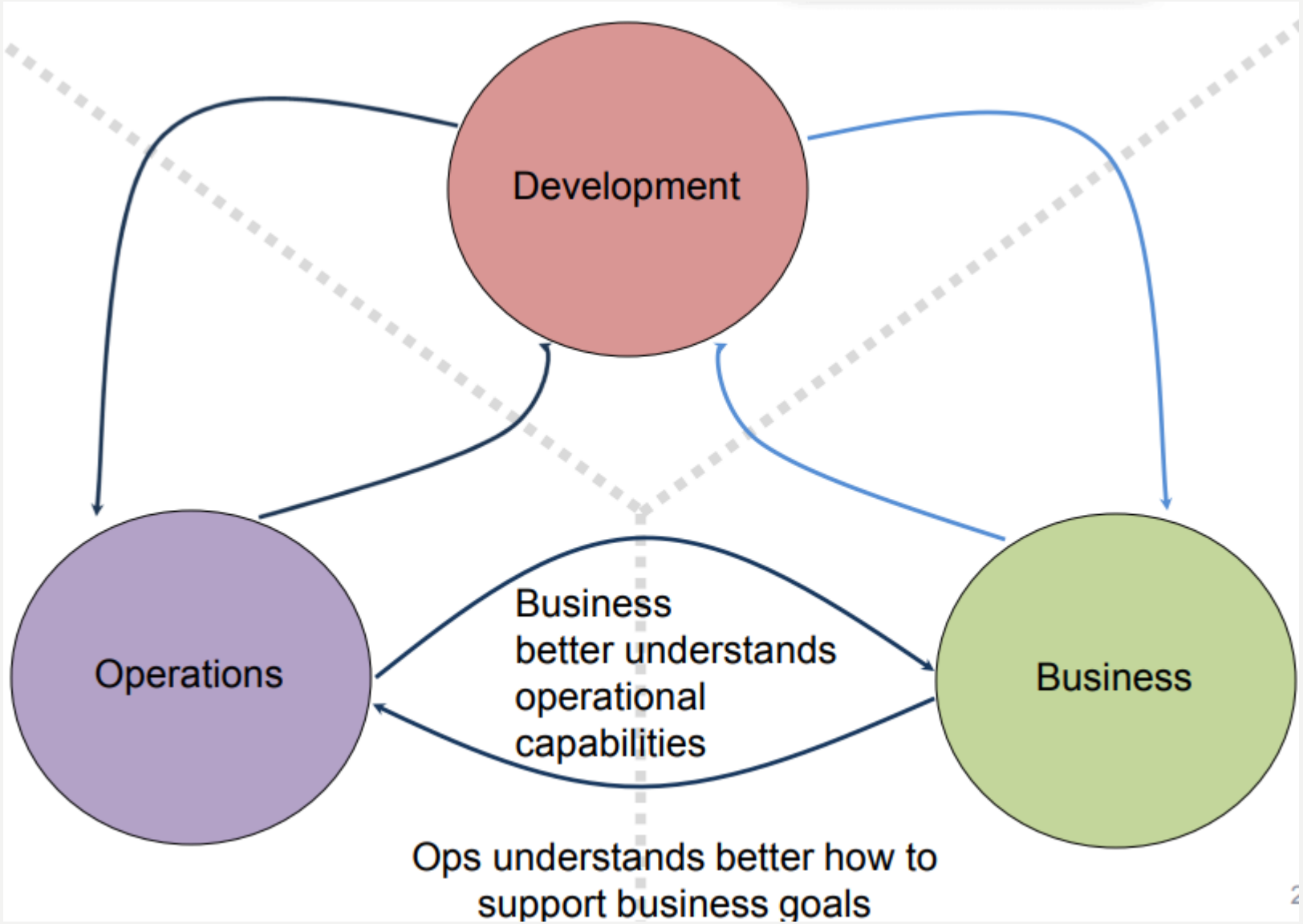


NOTE:

a

DevOps: Background

DevOps

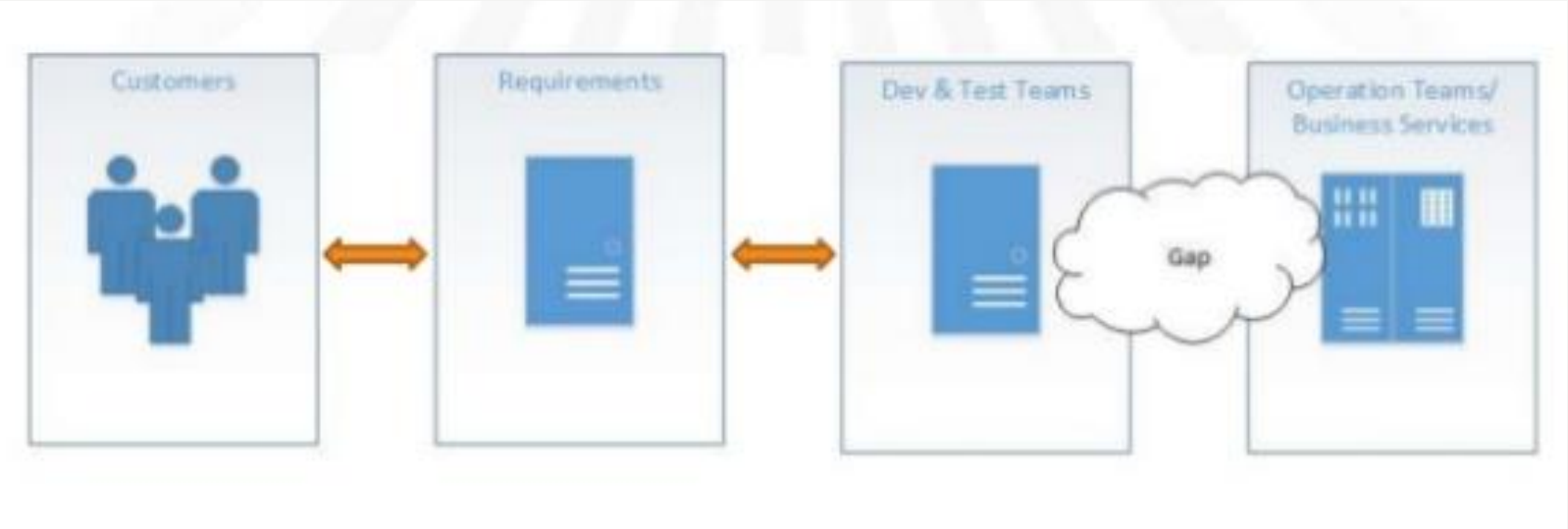


NOTE:

a

DevOps: Why ?

DevOps



NOTE:

a

DevOps: Top 3 Delivery Challenges

DevOps

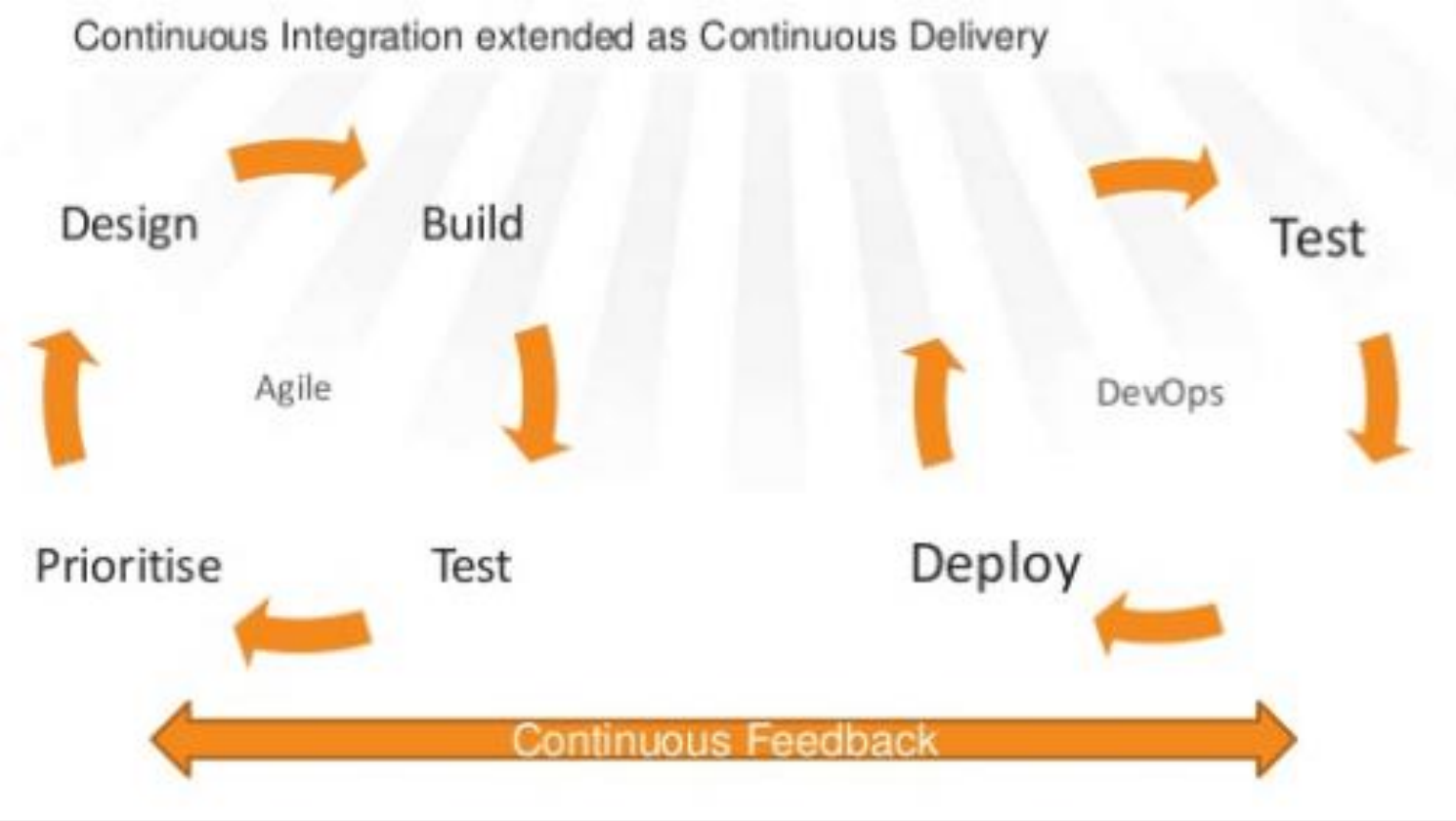
- Release management
  - Better understanding of risks, dependencies, compliance issues
- Release/Deployment coordination
  - Better tracking of discrete activities, faster escalation of issues, documented process control and granular reporting
- Release/Deployment Automation
  - Usually have existing automation but want to flexibly manage and drive this automation that can be invoked by non-operations resources in specific non-production environments

NOTE:

a

DevOps: Agile and DevOps

DevOps

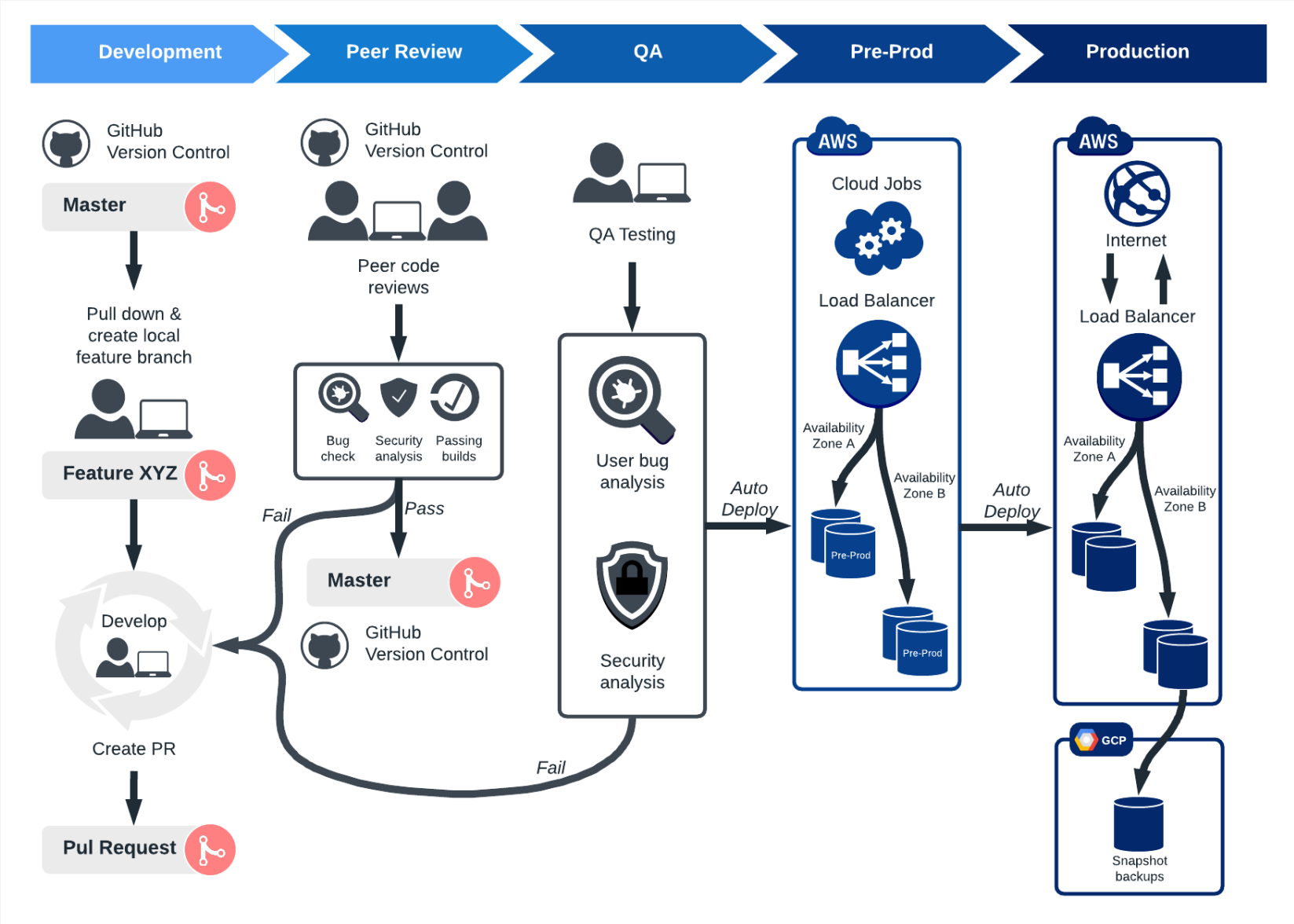


NOTE:

a

DevOps: Tools

DevOps

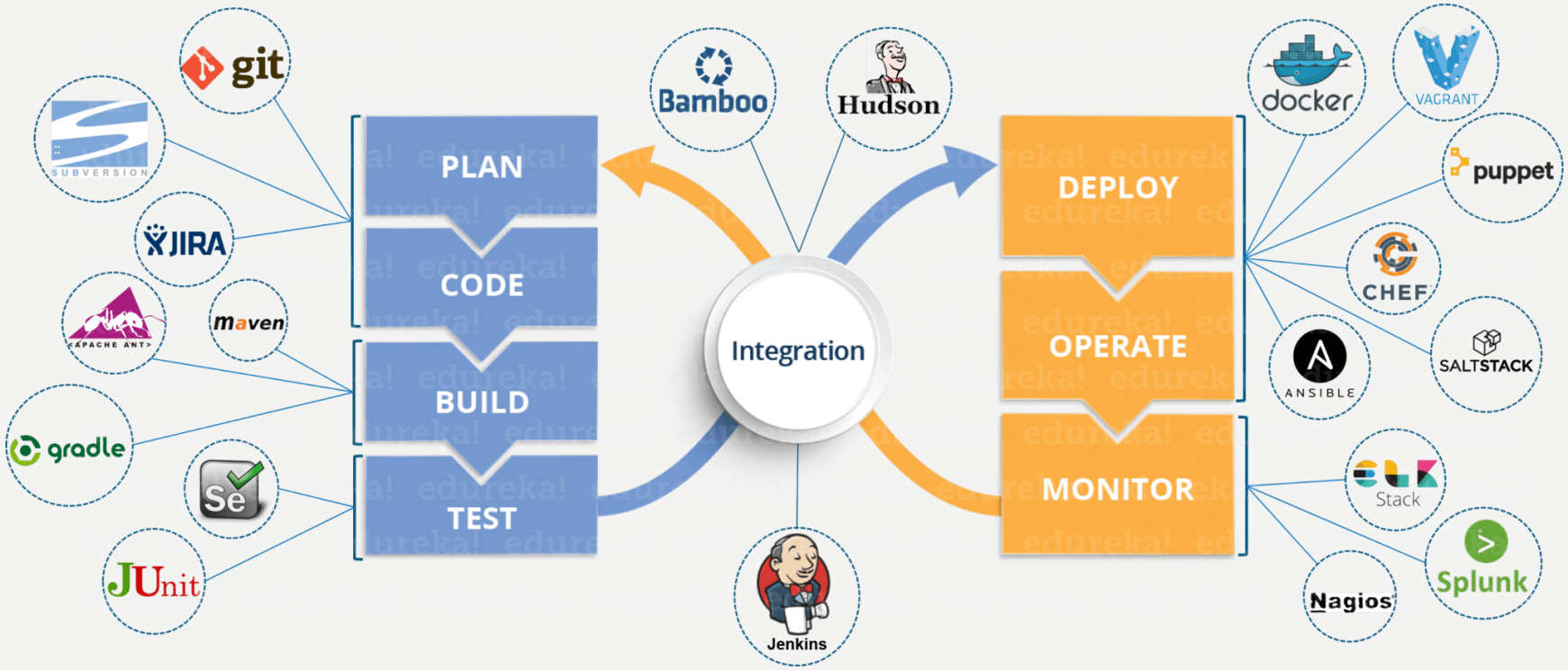


NOTE:

a

DevOps: Tools

DevOps

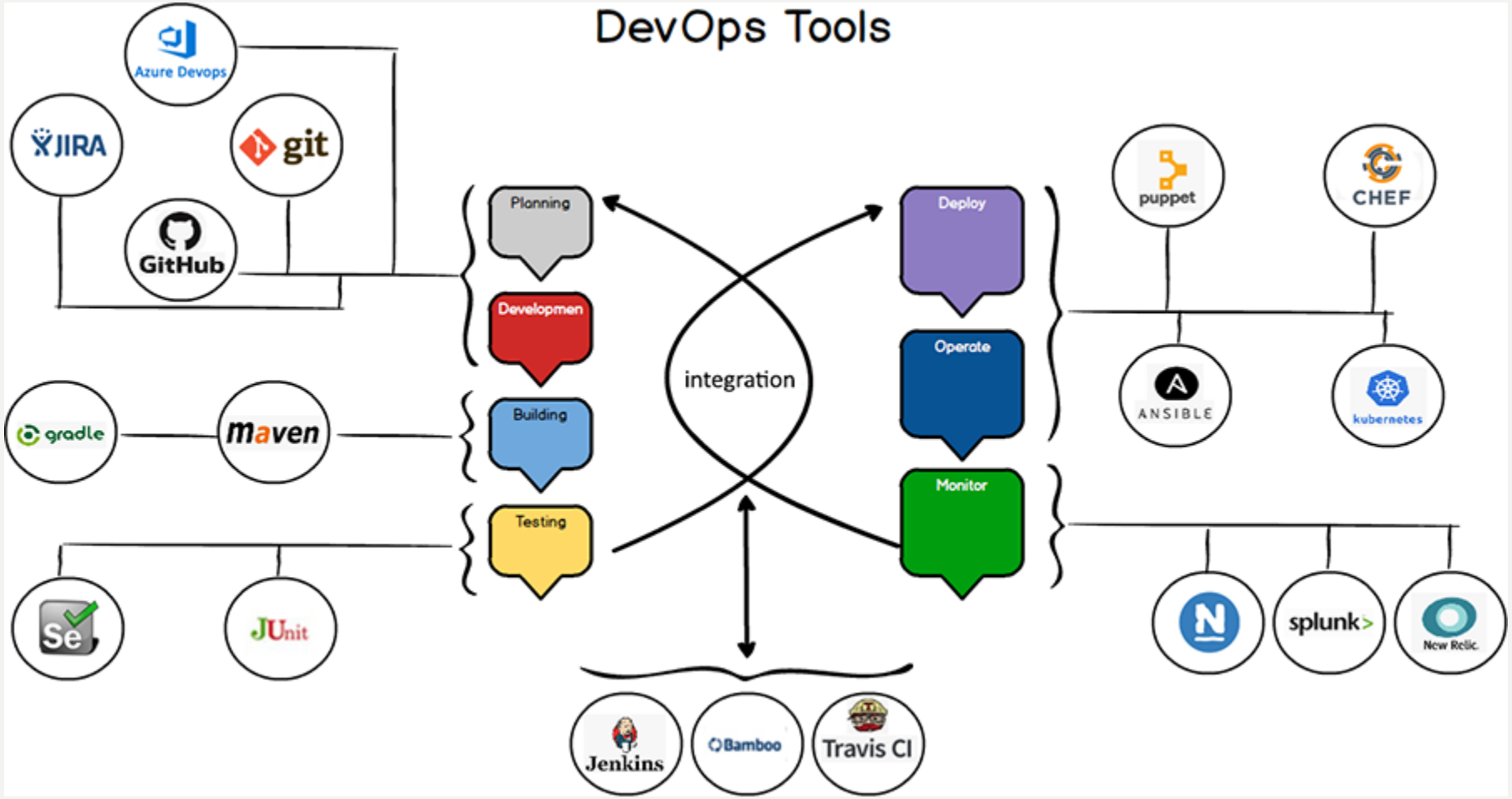


NOTE:

a

DevOps: Tools

DevOps



NOTE:

a

Git and GitHub - Introduction

Introduction

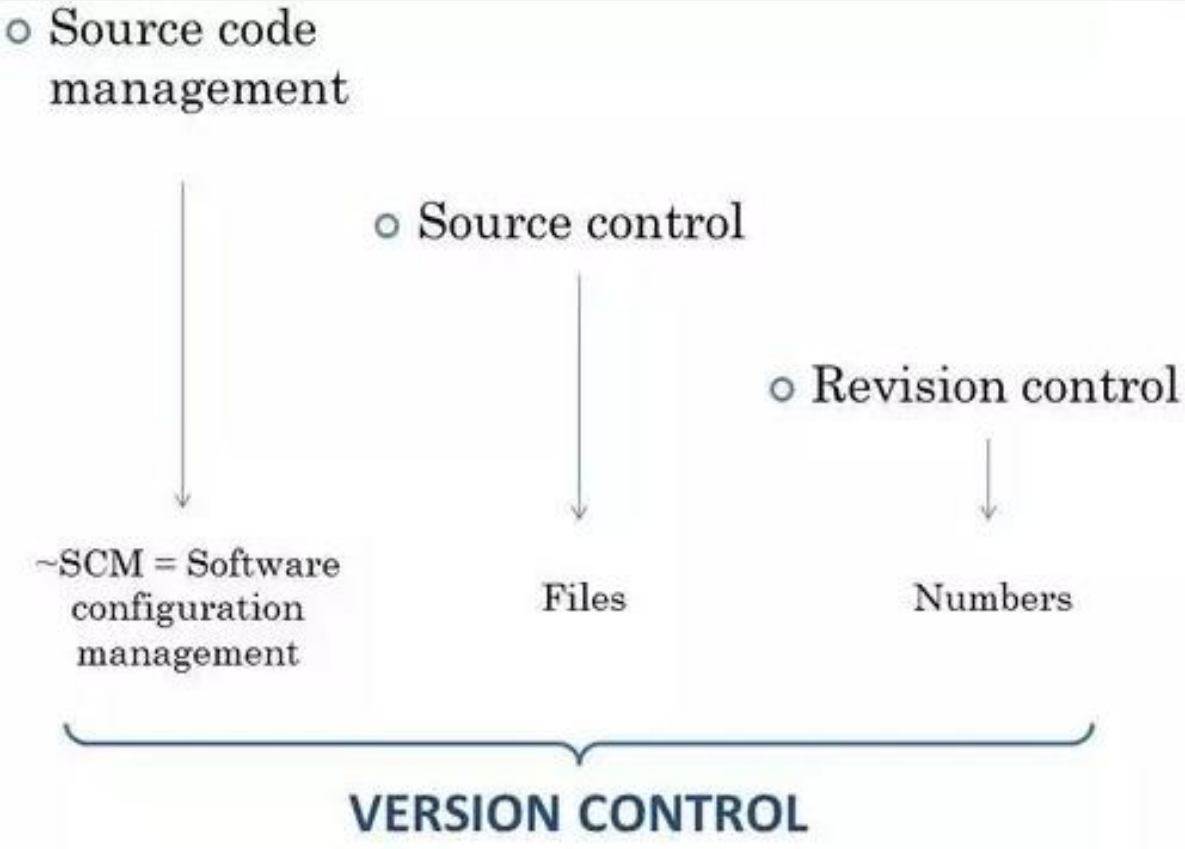
NOTE:

a

Git and GitHub - Introduction

What is a 'version control system?

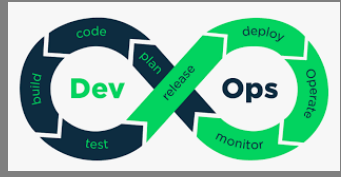
- a way to manage files and directories
- track changes over time
- recall previous versions
- 'source control' is a subset of a VCS.



NOTE:

a

Git and GitHub - History



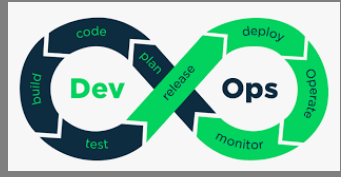
Some history of source control…

(1972) Source Code Control System (SCCS) - closed source, part of UNIX
(1982) Revision Control System(RCS)
- open source
(1986) Concurrent Versions System (CVS) - open source
(2000) Apache Subversion (SVN) - open source
(2000) BitKeeper SCM
- closed source, proprietary, used with source code management of Linux kernel
- free until 2005
- distributed version control

NOTE:

a

Git and GitHub - History



Some history of source control…

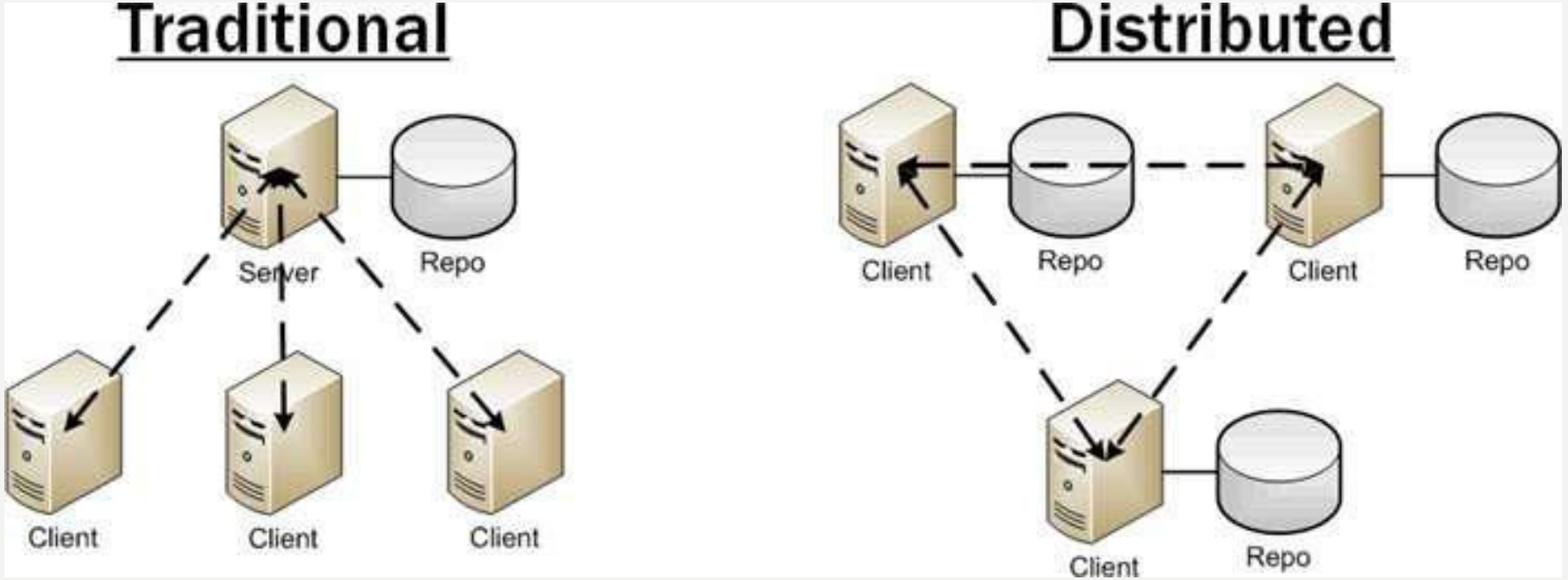
(1972) Source Code Control System (SCCS) - closed source, part of UNIX
(1982) Revision Control System(RCS)
- open source
(1986) Concurrent Versions System (CVS) - open source
(2000) Apache Subversion (SVN) - open source
(2000) BitKeeper SCM
- closed source, proprietary, used with source code management of Linux kernel
- free until 2005
- distributed version control

NOTE:

a

Git and GitHub - Distributed version control

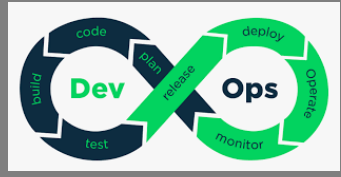
No central server
Every developer is a client, the server and the repository



NOTE:

a

Git



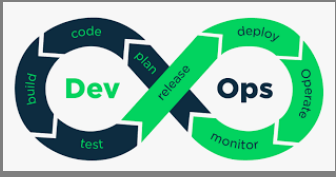
What is git?

- created by Linus Torvalds, April 2005
- replacement for BitKeeper to manage Linux kernel changes
- a command line version control program
- uses checksums to ensure data integrity
- distributed version control (like BitKeeper)
- cross-platform (including Windows!)
- open source, free

NOTE:

a

Git distributed version control



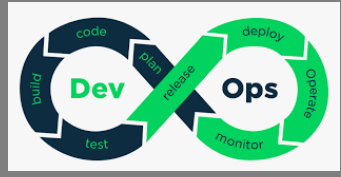
Git distributed version control

- "If we're not distributed, we're not worth using." – Linus Torvalds
- no need to connect to central server
- can work without internet connection
- no single failure point
- developers can work independently and merge their work later
- every copy of a Git repository can serve either as the server or as a client (and has complete history!)
- Git tracks changes, not versions
- Bunch of little change sets floating around

NOTE:

a

Git distributed version control



Is Git for me?

- People primarily working with source code
- Anyone wanting to track edits (especially changes to text files)
  - review history of changes
  - anyone wanting to share, merge changes
- Anyone not afraid of command line tools

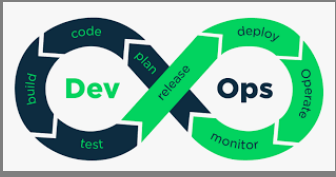
Most popular languages used with Git

• HTML • CSS • Javascript • Python • ASP • Scala • Shell scripts • PHP • Ruby • Ruby on Rails	• Perl • Java • C • C++ • C# • Objective C • Haskell • CoffeeScript • ActionScript

NOTE:

Not as useful for image, movies, music…and files that must be interpreted (.pdf, .psd, etc.)

Git distributed version control



What is a repository?

- "repo" = repository
- usually used to organize a single project
- repos can contain folders and files, images, videos, spreadsheets, and data sets – anything our project needs

NOTE:

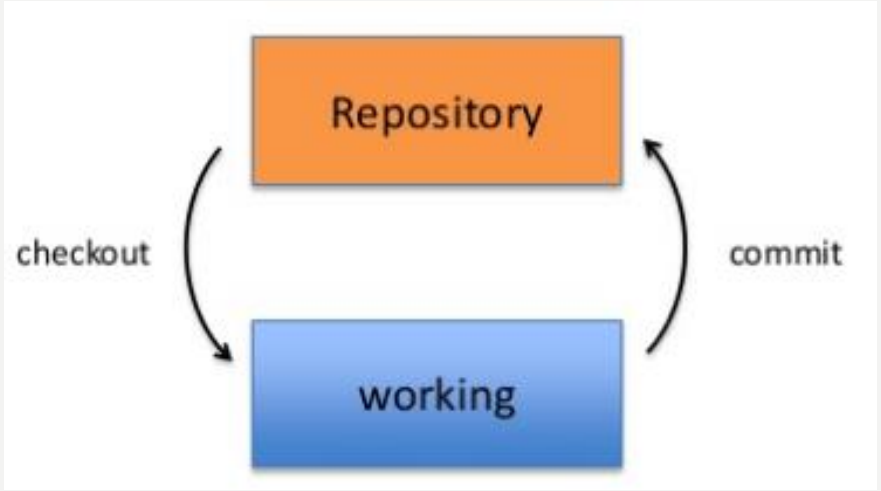
a

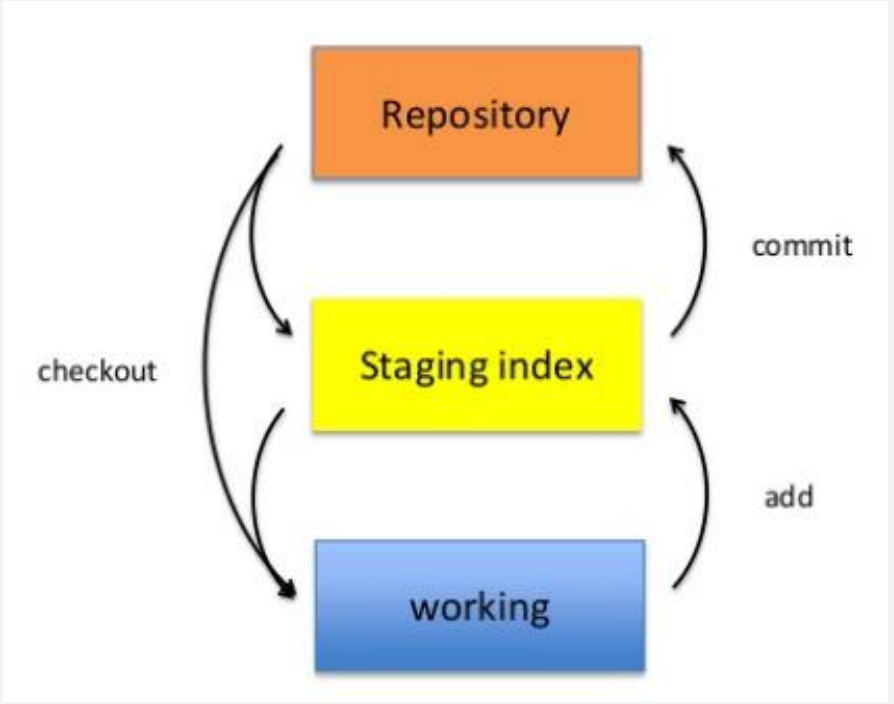
What is a repository?

- "repo" = repository
- usually used to organize a single project
- repos can contain folders and files, images, videos, spreadsheets, and data sets – anything our project needs

Git distributed version control

Two Tier vs Three Tier





NOTE:

a

Git :A simple Git workflow

A simple Git workflow

1.Initialize a new project in a directory: git init

```
[ dolanmi L02029756  ~/Desktop ]$ mkdir new_project
[ dolanmi L02029756  ~/Desktop ]$ cd new_project/
[ dolanmi L02029756  ~/Desktop/new_project ]$ git init
Initialized empty Git repository in /Users/dolanmi/Desktop/new_project/.git/
[ dolanmi L02029756  ~/Desktop/new_project ]$
```

2.Add a file using a text editor to the directory

3.Add every change that has been made to the directory: **git add.**

4.Commit the change to the repo: **git commit –m "important message here"**

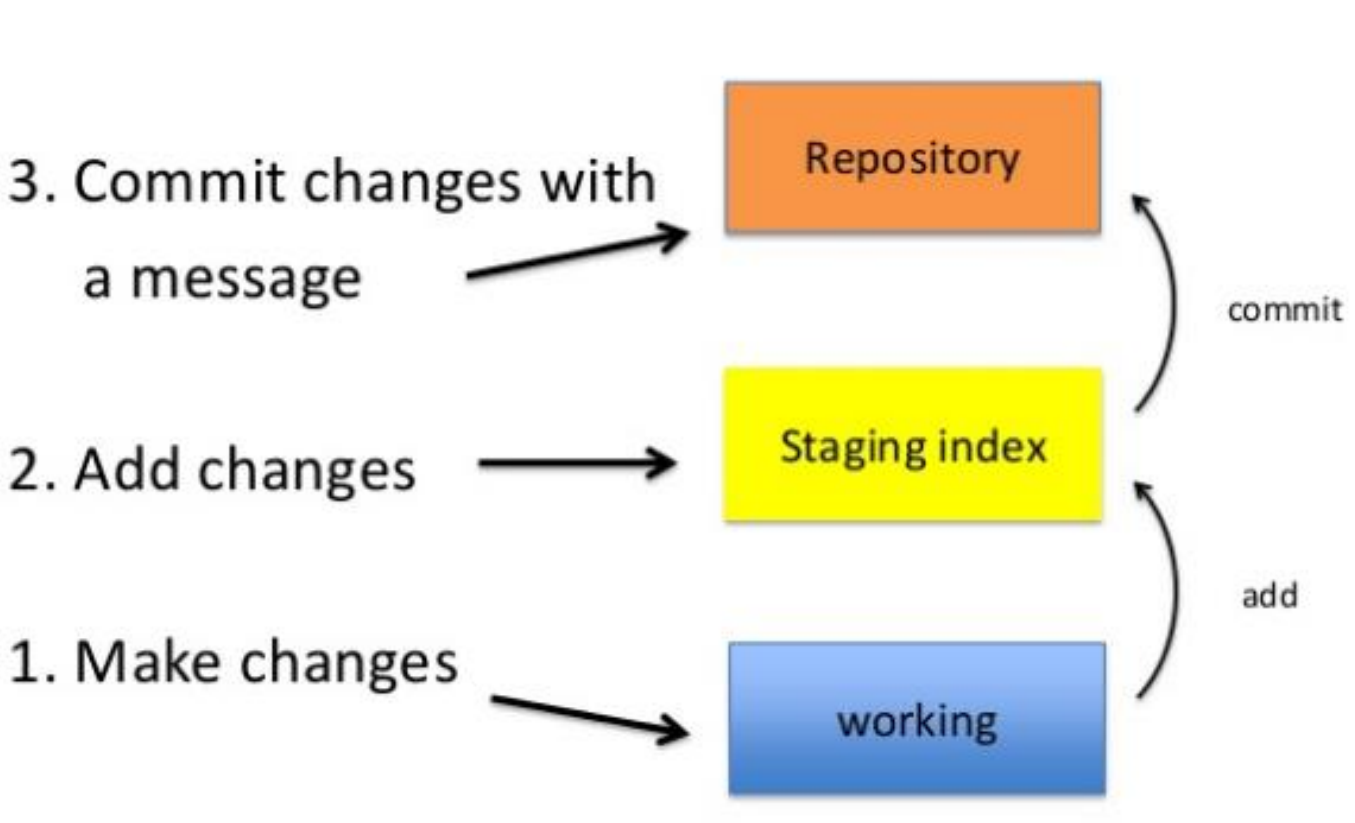
```
[ dolanmi L02029756  ~/Desktop/new_project ]$ git add .
[ dolanmi L02029756  ~/Desktop/new_project ]$ git commit -m "Add message to file.txt"
[master (root-commit) 1a7e4a5] Add message to file.txt
 1 file changed, 1 insertion(+)
 create mode 100644 file.txt
[ dolanmi L02029756  ~/Desktop/new_project ]$
```

NOTE:

a

Git distributed version control

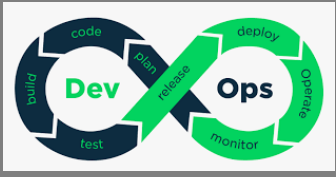
After initializing a new git repo…



NOTE:

a

Git distributed version control



A note about commit messages

- Tell what it does (present tense)
- Single line summary followed by blank space followed by more complete description
- Keep lines to <= 72 characters
- Ticket or bug number helps

Good and bad examples

Bad: “Typo fix”

Good: “Add missing / in CSS section”

Bad: “Updates the table. We’ll discuss next Monday with Darrell.”

Bad: git commit -m "Fix login bug”

Good: git commit -m

```
Redirect user to the requested page after login

https://trello.com/path/to/relevant/card

Users were being redirected to the home page after login, which is less
useful than redirecting to the page they had originally requested before
being redirected to the login form.

* Store requested path in a session variable
* Redirect to the stored location after successfully logging in the user
```

NOTE:

a

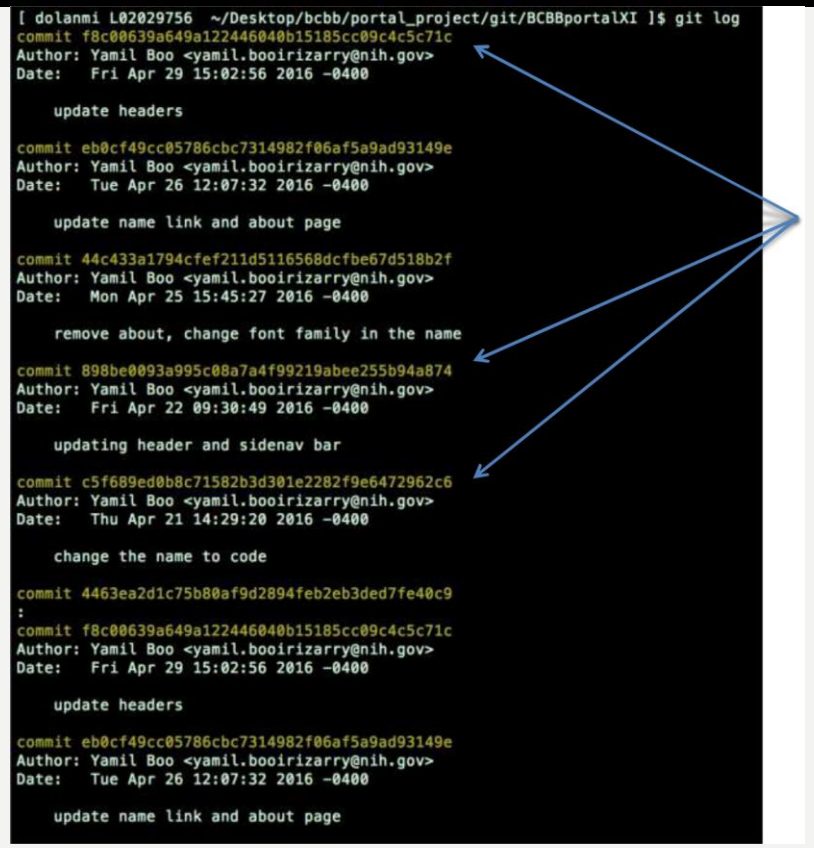
Git distributed version control

How to I see what was done?

git log

```
[ dolanmi L02029756  ~/Desktop/new_project ]$ git log
commit 6c40ffd9ba4ba1567eb6fcd3715f12a15b0a678d
Author: mchldln <dolanmi@niaid.nih.gov>
Date:   Mon May 2 18:11:23 2016 -0400

    Add message to text file
[ dolanmi L02029756  ~/Desktop/new_project ]$
```



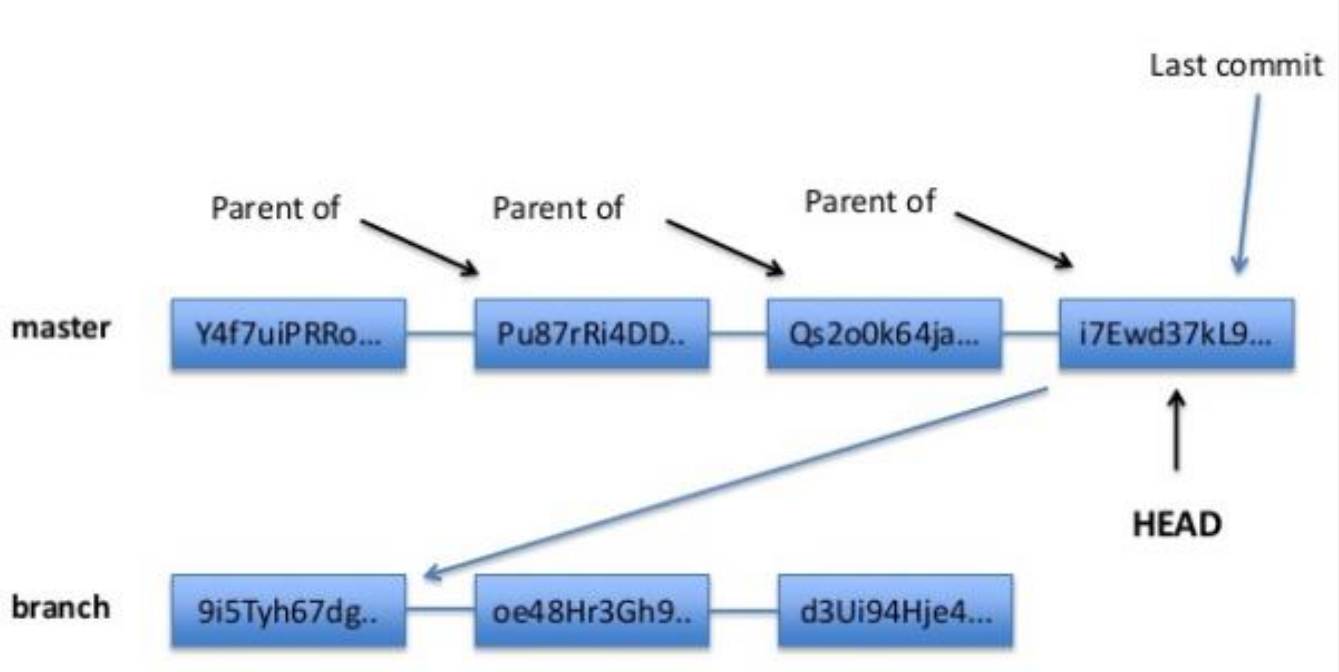
NOTE:

a

Git distributed version control

The HEAD pointer

- points to a specific commit in repo
- as new commits are made, the pointer changes
- HEAD always points to the "tip" of the currently checked-out branch in the repo
- (not the working directory or staging index)
- last state of repo (what was checked out initially)
- HEAD points to parent of next commit (where writing the next commit takes place)



NOTE:

a

Git distributed version control

Which files were changed and where do they sit in the three tree?

git status – allows one to see where files are in the three tree scheme

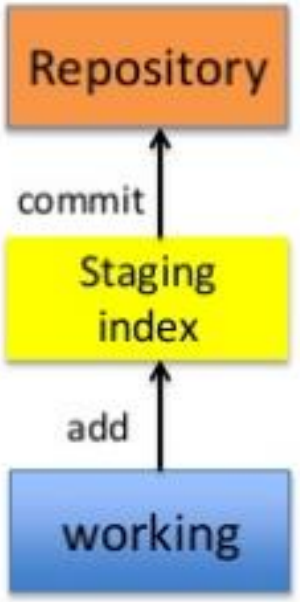
```
[ dolanmi L02029756  ~/Desktop/new_project ]$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   file.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

```
[ dolanmi L02029756  ~/Desktop/new_project ]$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   file.txt
```



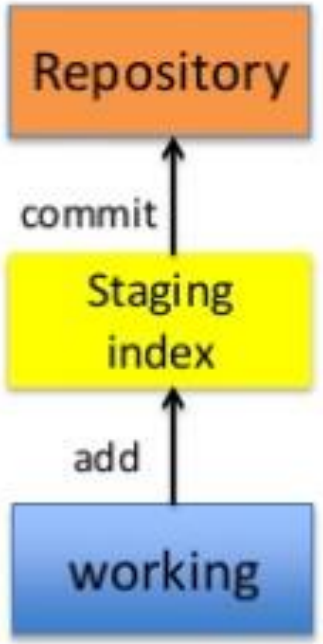
NOTE:

a

Git distributed version control

What changed in working directory?

git diff – compares changes to files between repo and working directory



Note: git diff --staged - compares staging index to repo

Note: git diff filename can be used as well

NOTE:

a

Git distributed version control

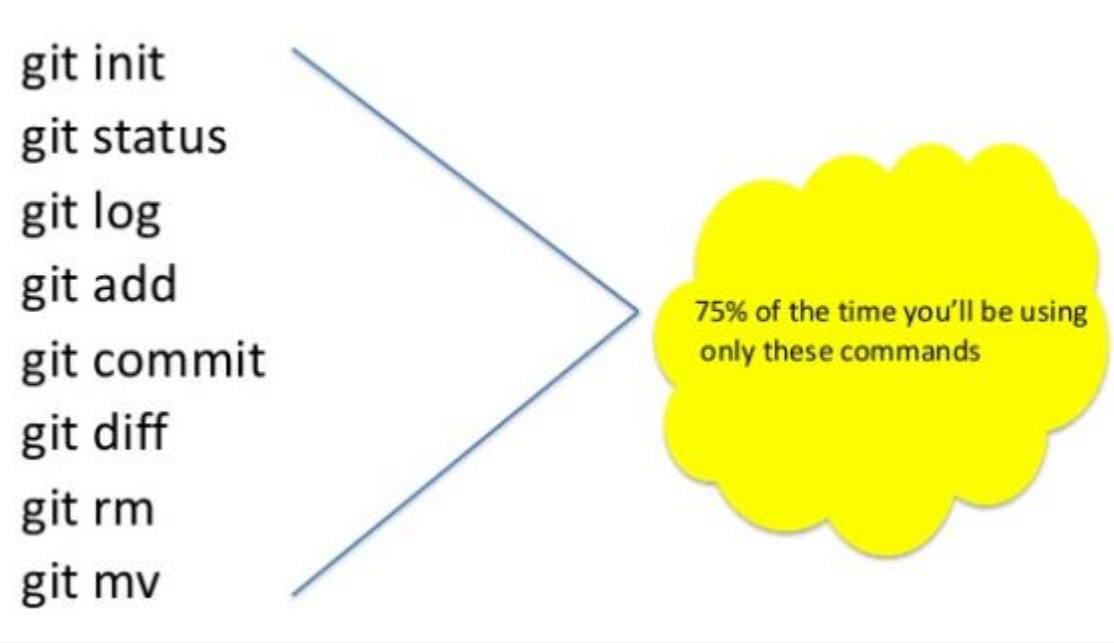
Deleting files from the repo

git rm filename.txt

- moves deleted file change to staging area
- It is not enough to delete the file in our working directory. we must commit the change.

Moving (renaming) files

git mv filename1.txt filename2.txt



NOTE:

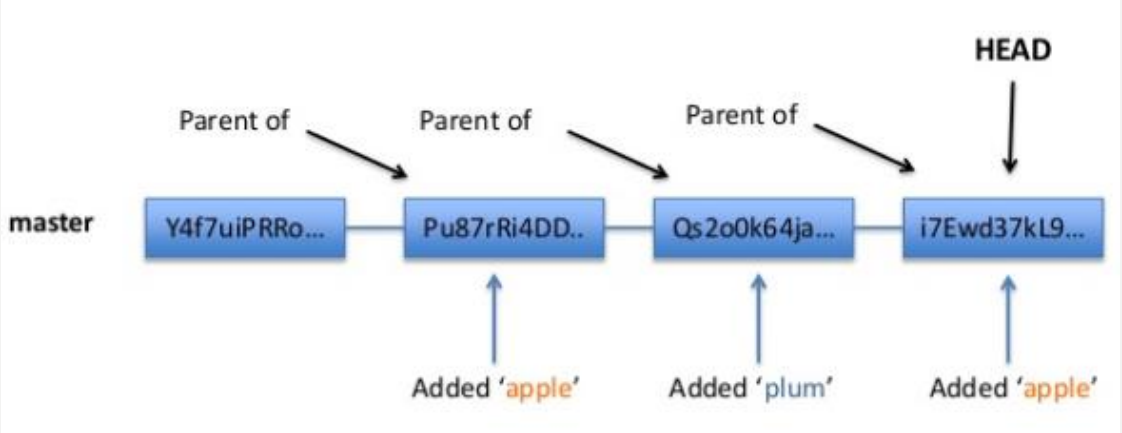
a

Git distributed version control

What if I want to undo changes made to working directory?
git checkout something
(where "something" is a file or an entire branch) Repository
git checkout will grab the file from the repo
• Example: git checkout -- file1.txt

What if I want to undo changes added to staging area?
git reset HEAD filename.txt
What if I want to undo changes committed to the repo?
git commit --amend -m "message"
- allows one to amend a change to the last commit
- anything in staging area will be amended to the last commit

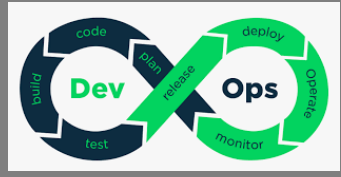
Note: To undo changes to older commits, make a new commit



NOTE:

a

Git distributed version control



Obtain older versions

git checkout 6e073c640928b -- filename.txt

Note: Checking out older commits places them into the staging area

Which files are in a repo?

git ls-tree tree-ish

tree-ish – a way to reference a repo full SHA, part SHA, HEAD, others

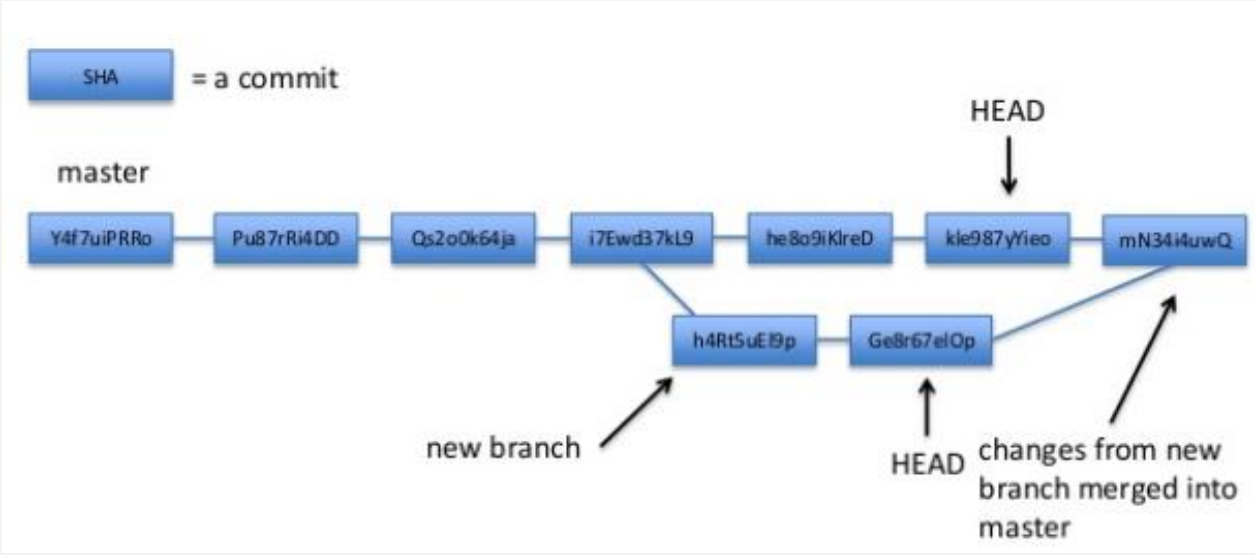
NOTE:

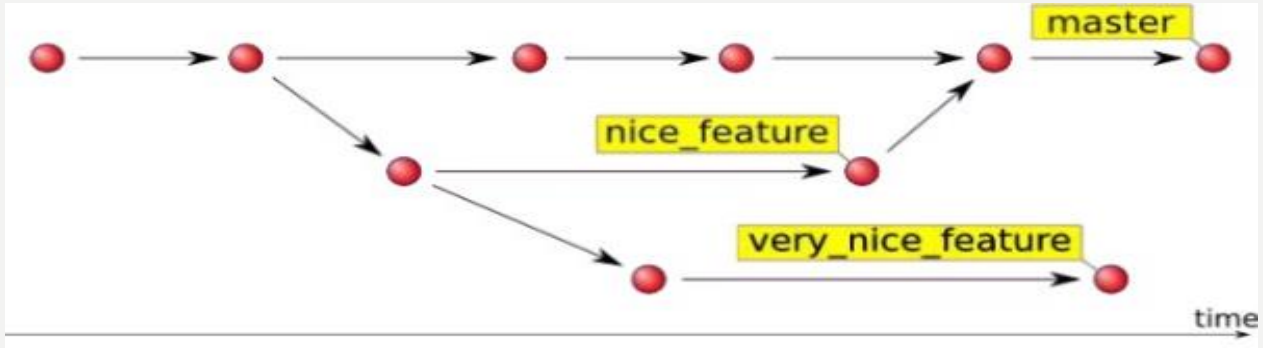
a

Git distributed version control

branching

- allows one to try new ideas
- If an idea doesn't work, throw away the branch. Don't have to undo many changes to master branch
- If it does work, merge ideas into master branch.
- There is only one working directory

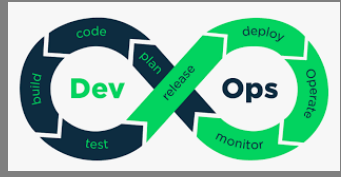




NOTE:

a

Git distributed version control



In which branch am I?
git branch

How do I create a new branch?
git branch new_branch_name
Note: At this point, both HEADs of the branches are pointing to the same commit (that of master)

How do I switch to new branch?
git checkout new_branch_name
At this point, one can switch between branches, making commits, etc. in either branch, while the two stay separate from one another.

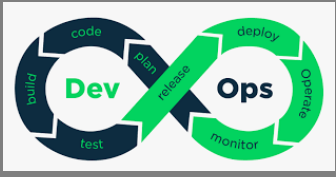
Note: In order to switch to another branch, our current working directory must be clean (no conflicts, resulting in data loss).

Comparing branches
git diff first_branch..second_branch

NOTE:

a

Git distributed version control



How do I merge a branch?
From the branch into which we want to merge another branch….
git merge branch_to_merge
Note: Always have a clean working directory when merging

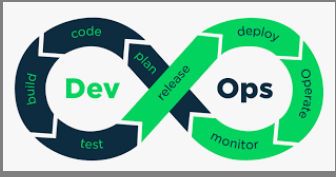
“fast-forward” merge occurs when HEAD of master

branch is seen when looking back

NOTE:

a

Git – Pull Request vs Push Request



Pull requests display diffs to compare the changes we made in our topic branch against the base branch that we want to merge our changes into.

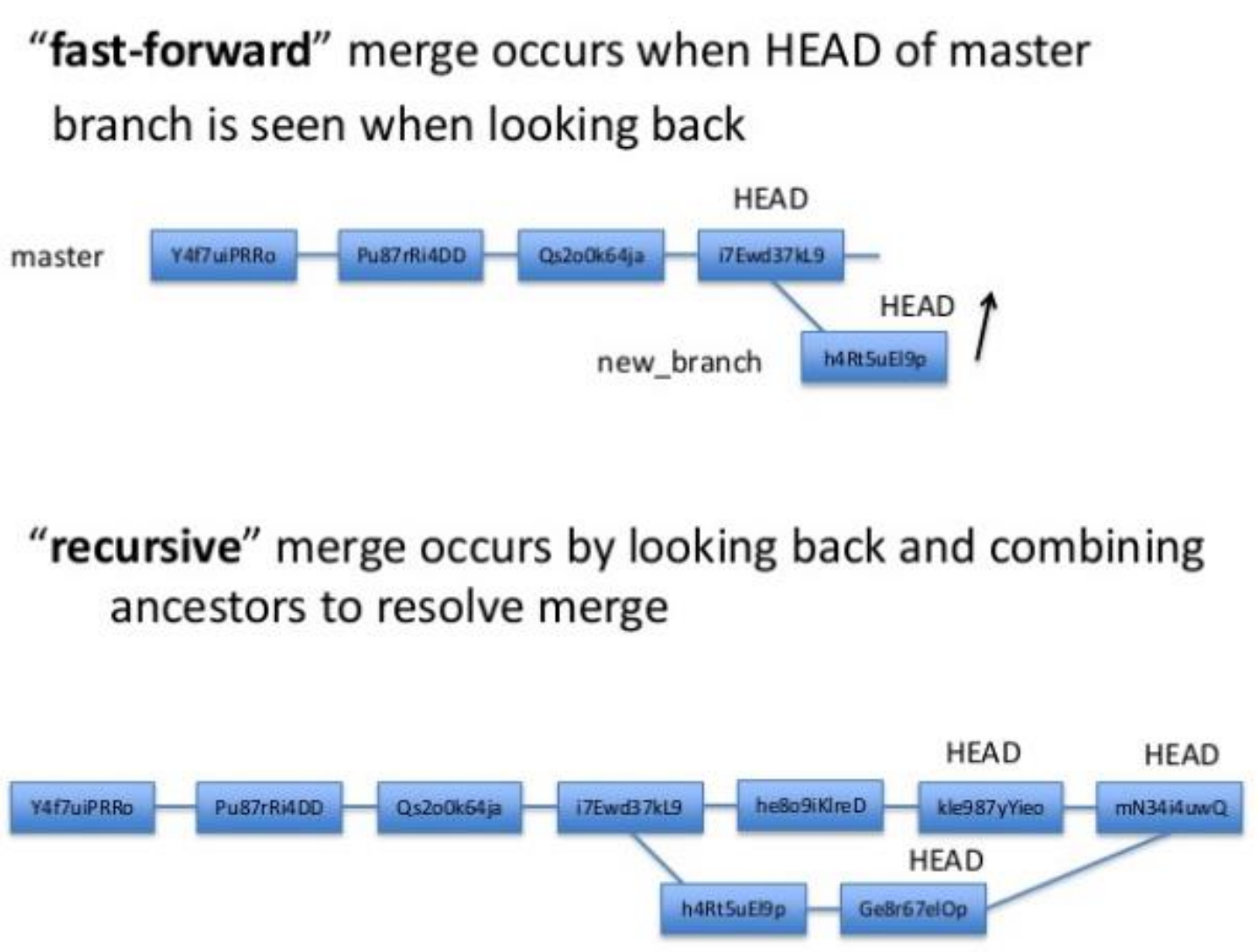
While pull requests are not a core feature of Git, they are commonplace when it comes to collaborating with Git hosting services.
They are especially necessary when working with open-source projects. ... Most open-source projects have a maintainer who can control which changes are approved and merged into the project.

NOTE:

a

Git distributed version control

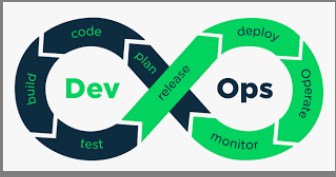
A



NOTE:

a

Git distributed version control



merge conflicts

What if there are two changes to same line in two different commits?

Resolving merge conflicts
Git will notate the conflict in the files!
Solutions:

1. Abort the merge using git merge –abort
2. Manually fix the conflict
3. Use a merge tool (there are many out there)

Graphing merge history
git log --graph --oneline --all --decorate

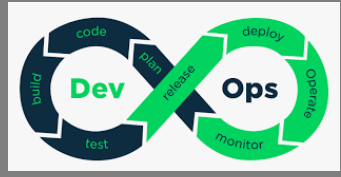
Tips to reduce merge pain

- merge often
- keep commits small/focused
- bring changes occurring to master into our branch frequently (“tracking”)

NOTE:

a

GitHub



GitHub

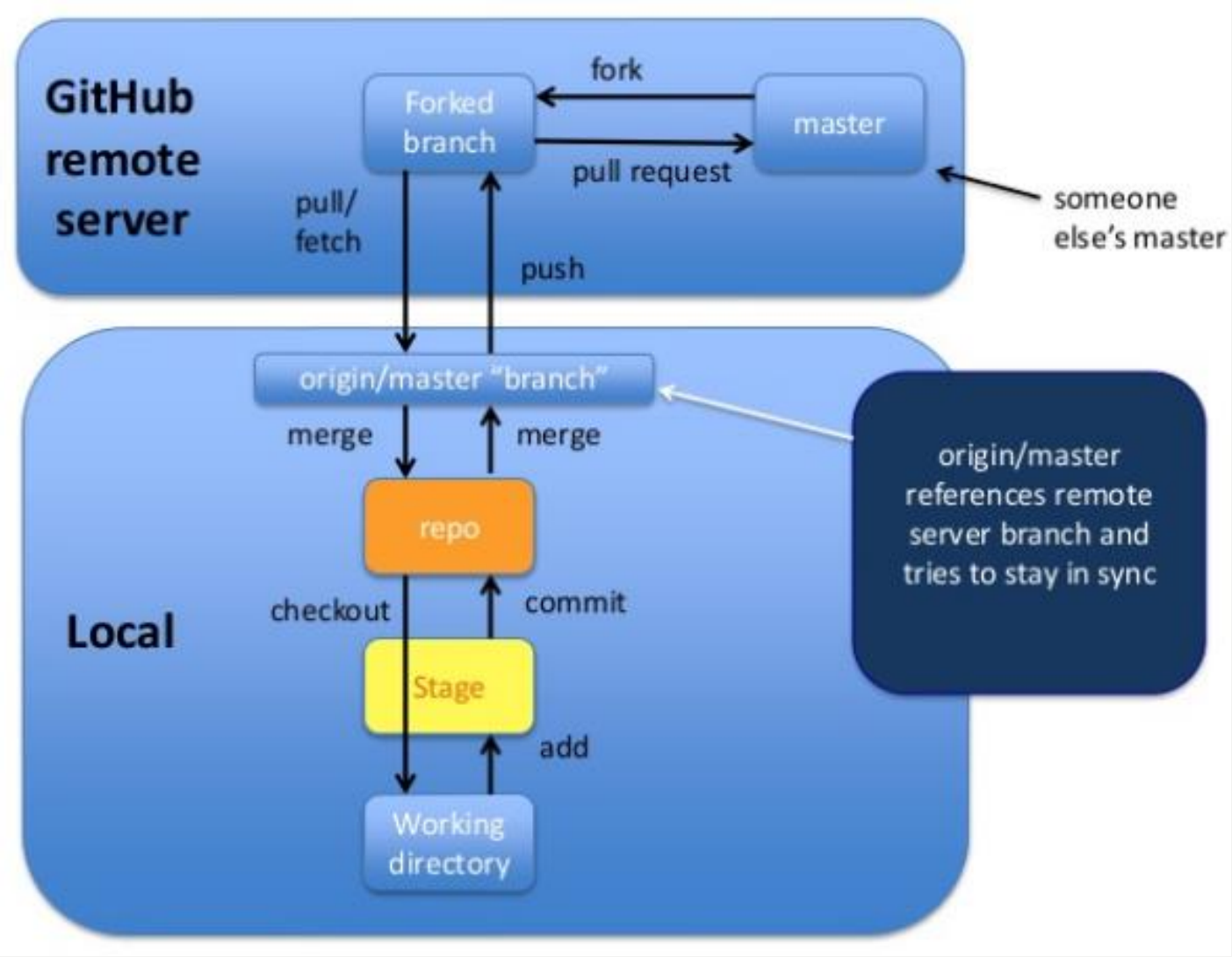
- a platform to host git code repositories
- http://github.com
- launched in 2008
- most popular Git host
- allows users to collaborate on projects from anywhere
- GitHub makes git social!
- Free to start

NOTE:

a

GitHub

Sometimes developers choose to place repo on GitHub as a centralized place where everyone commits changes, but it doesn't have to be on GitHub

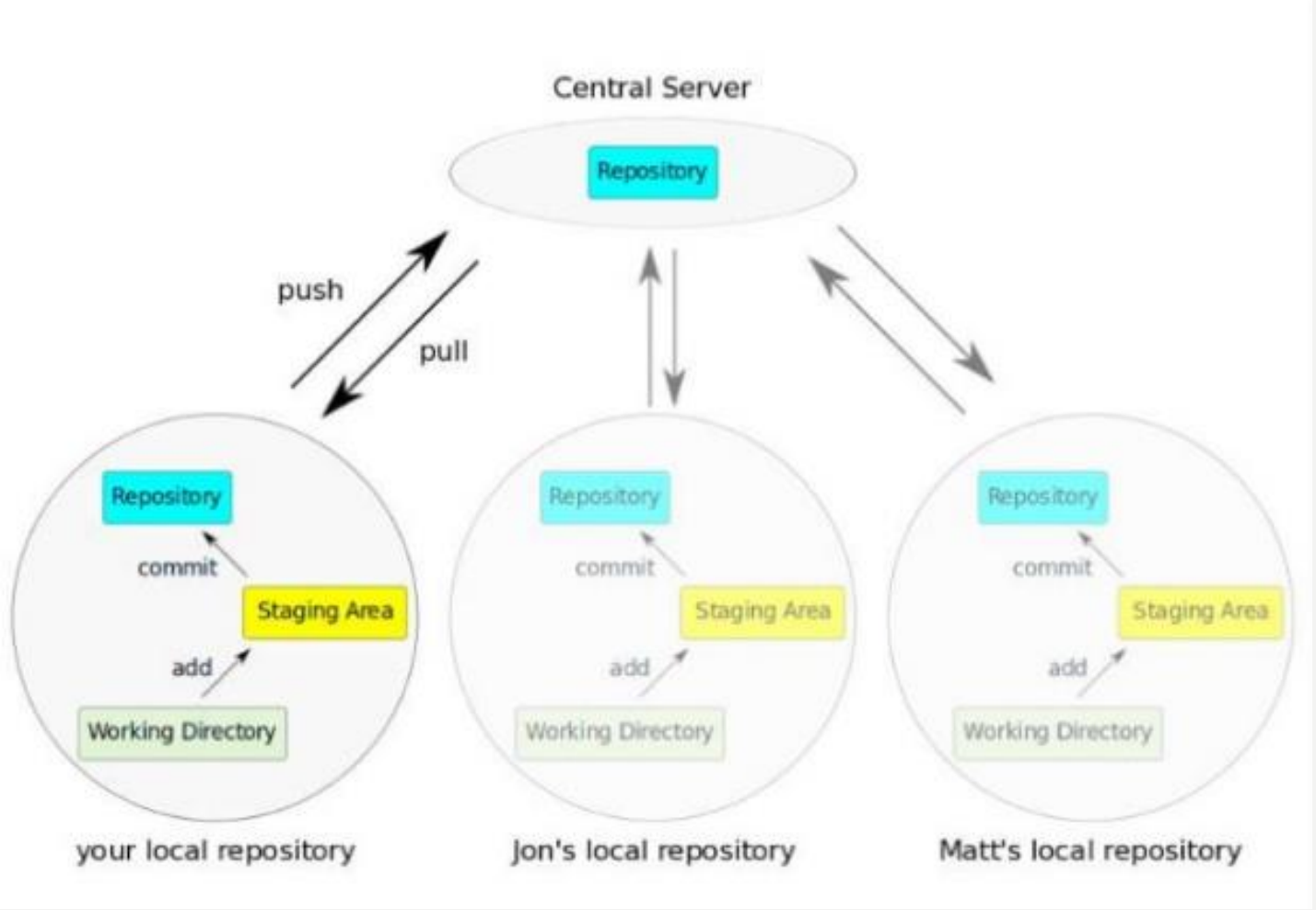


NOTE:

a

GitHub

Sometimes developers choose to place repo on GitHub as a centralized place where everyone commits changes, but it doesn't have to be on GitHub



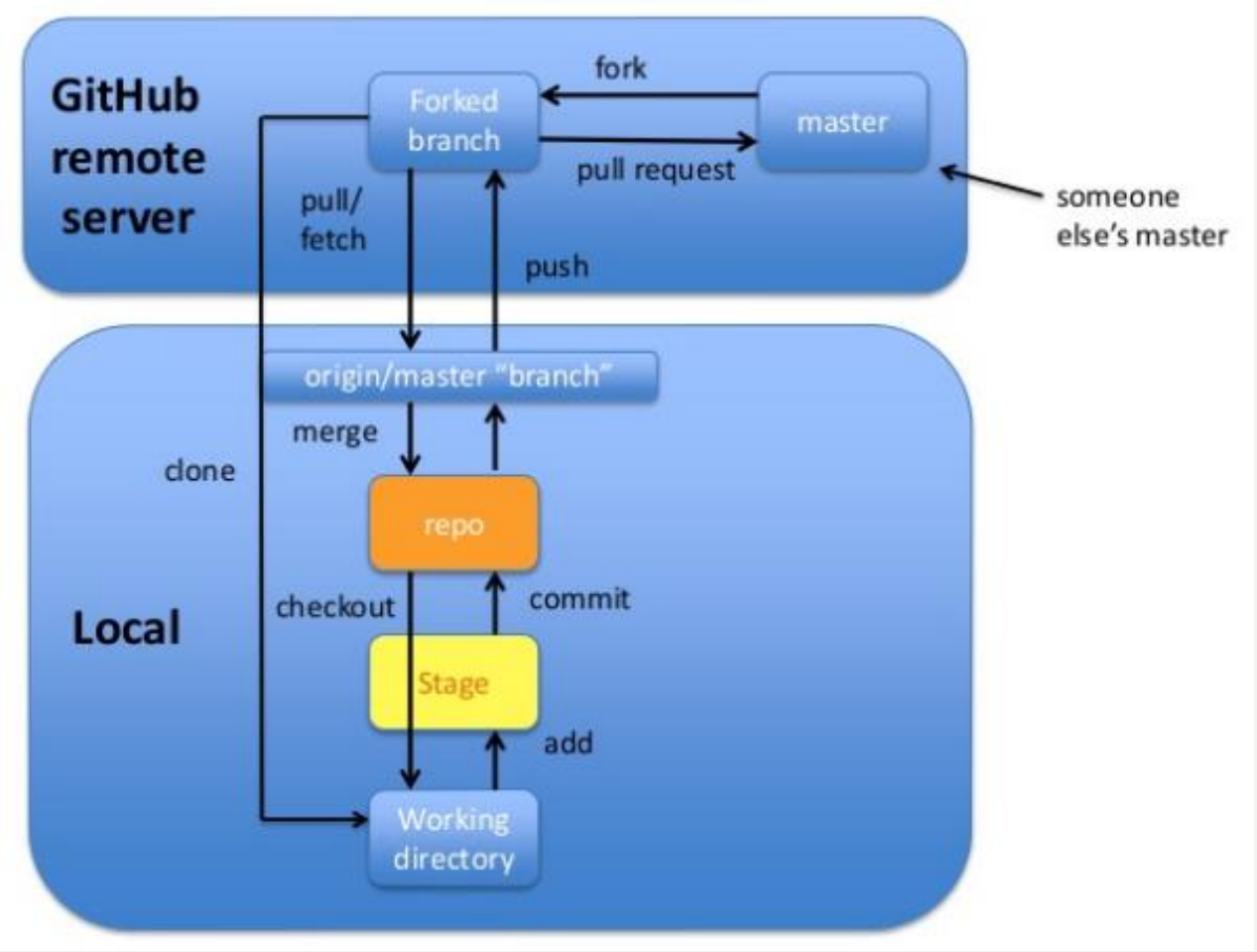
NOTE:

a

GitHub

Copying (cloning) files from remote repo to local machine

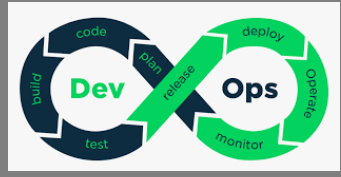
git clone URL <new_dir_name>



NOTE:

a

GitHub



How do I link my local repo to a remote repo?
git remote add <alias> <URL>

Note: This just establishes a connection…no files are copied/moved
Note: Yes! we may have more than one remote linked to our local directory!

Which remotes am I linked to?
git remote
Pushing to a remote repo

git push local_branch_alias branch_name
Fetching from a remote repo

git fetch remote_repo_name

Fetch in no way changes a our working dir or any commits that we've made.

- Fetch before we work
- Fetch before we push
- Fetch often

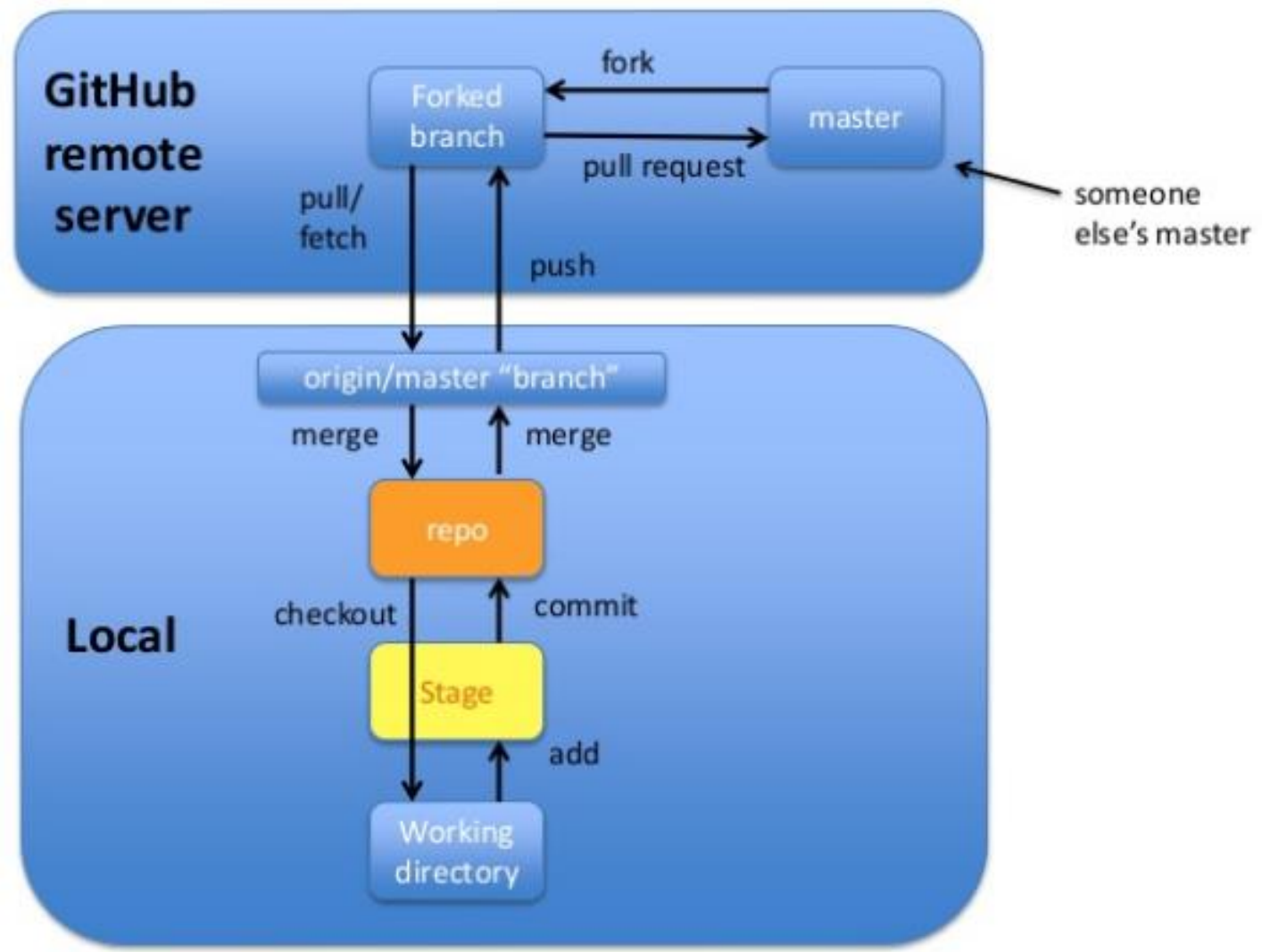
git merge must be done to merge fetched changes into local branch

NOTE:

a

GitHub

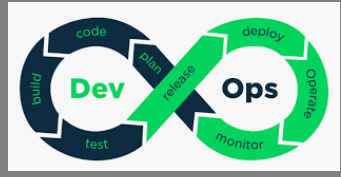
Collaborating with Git



NOTE:

a

GitHub



GitHub Gist

https://gist.github.com/

Good resources

- Git from Git:https://git-scm.com/book/en/v2
- A guided tour that walks through the fundamentals of Git: https://githowto.com
- Linus Torvalds on Git:

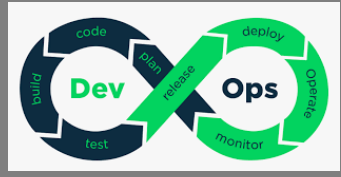
https://www.wetube.com/watch?v=idLyobOhtO4

- Git tutorial from Atlassian: https://www.atlassian.com/git/tutorials/
- A number of easy-to-understand guides by the GitHub folks https://guides.github.com

NOTE:

a

GitHub



git commit -a

- Allows one to add to staging index and commit at the same time
- Grabs everything in working direcotry
- Files not tracked or being deleted are not included

git log --oneline
- gets first line and checksum of all commits in current branch

git diff g5iU0oPe7x
When using checksum of older commit, will show we all changes compared to those in our working directory

Renaming and deleting branches
git branch –m/--move old_name new_name
git branch –d branch_name

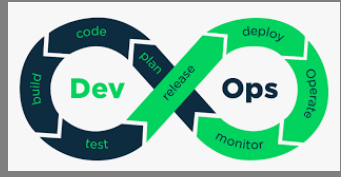
Note: Must not be in branch_name
Note: Must not have commits in branch_name unmerged in branch from which we are deleting

git branch –D branch_name

NOTE:

a

GitHub



Note : If we are *really* sure that we want to delete branch with commits

Tagging
- Git has the ability to tag specific points in history as being important, such as releases versions

(v.1.0, 2.0, …)
git tag

Tagging
Two types of tags:
lightweight – a pointer to a specific comment – basically a SHA stored in a file

git tag tag_name

annotated – a full object stored in the Git database – SHA, tagger name, email, date, message and can be signed and verified with GNU Privacy Guard (GPG)
git tag –a tag_name –m "message"

How do I see tags?

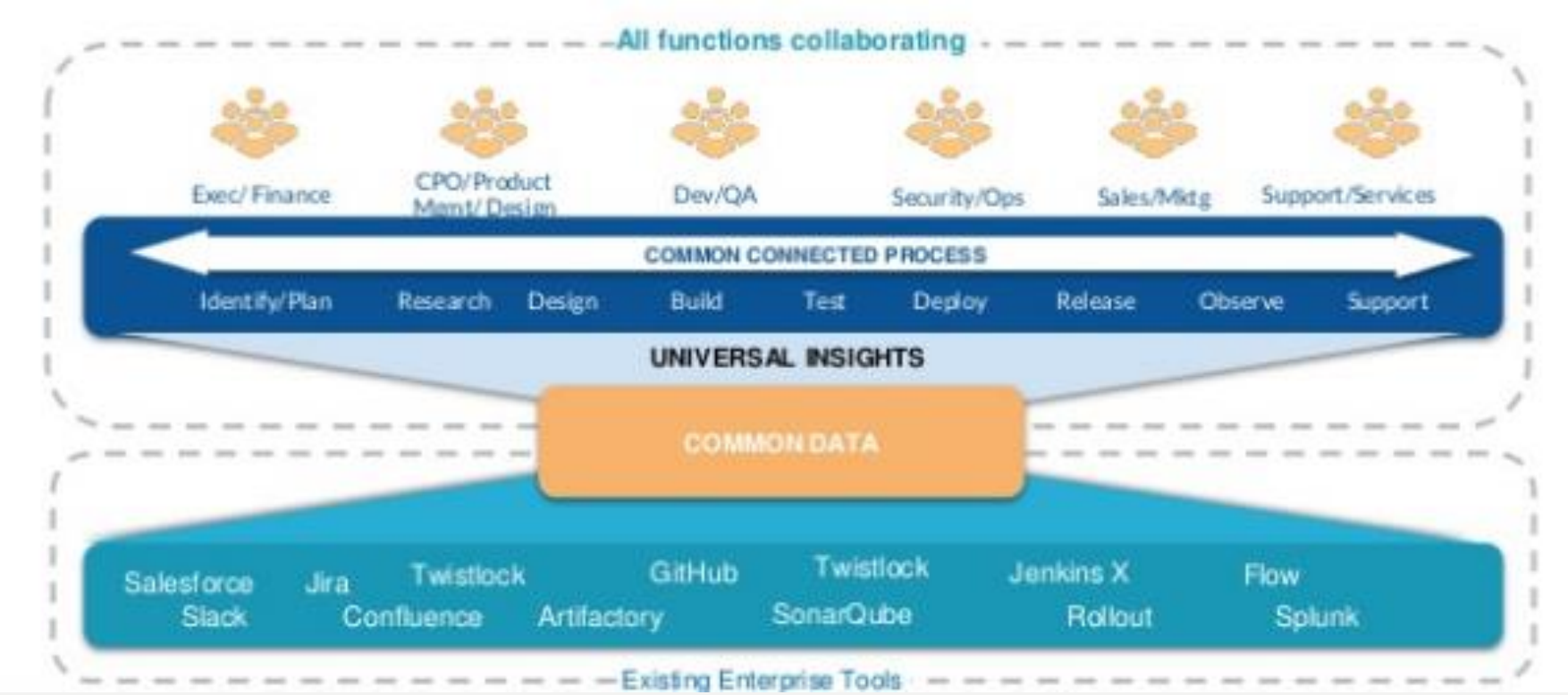
git show tag_name

NOTE:

a

Software Delivery Management

a

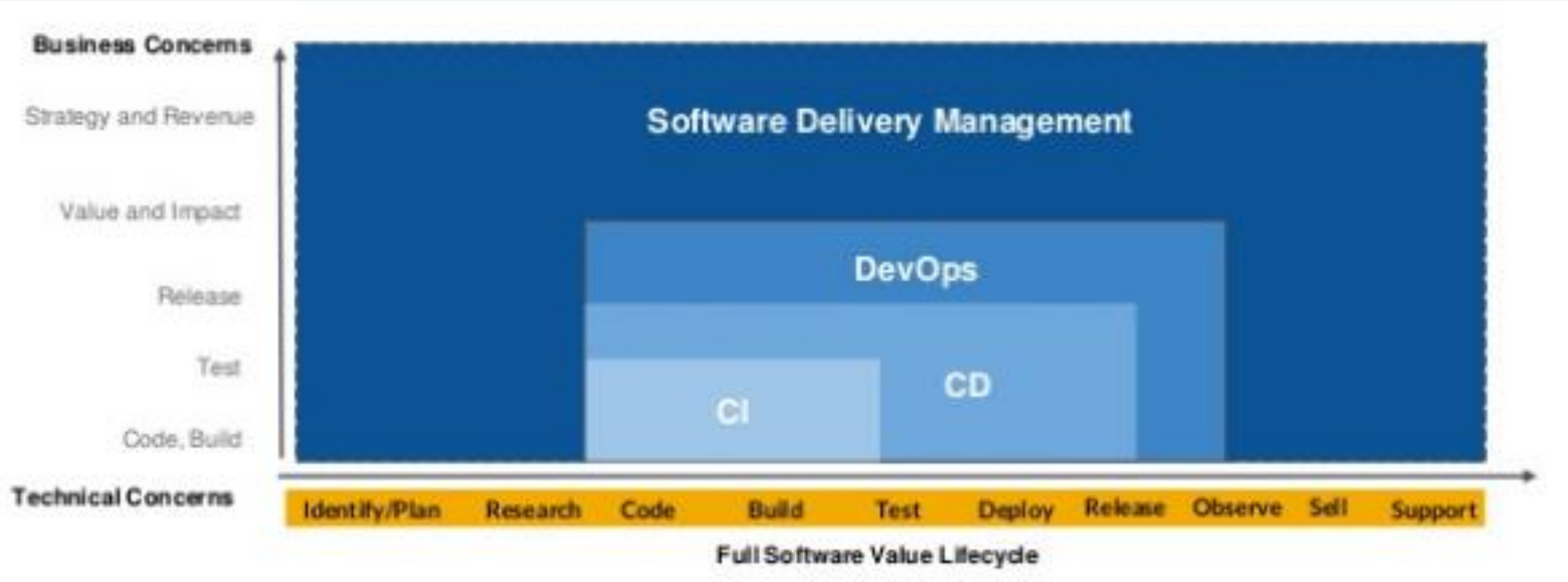


NOTE:

a

Software Delivery Management

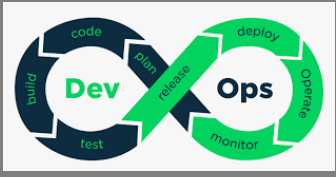
a



NOTE:

a

Software Delivery Management



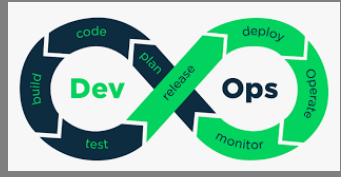
What is CI and CD?

- Continuous Integration
    - An approach to be continually validating the state of a codebase through automated testing.
    - Best achieved through integration with version control
- Continuous Delivery / Deployment
    - An approach to regularly deploying artifacts that successfully pass the CI phase to ensure confidence around the deployment

NOTE:

a

CI/CD



Continuous integration, continuous deployment, and continuous delivery are like vectors that have the same direction, but different magnitude.

Their goal is the same: make our software development and release process faster and more robust.

The key difference between the three is in the scope of automation applied.

Delivery vs Deployment

- Continuous Delivery
  - Automatically prepare and track a release to production

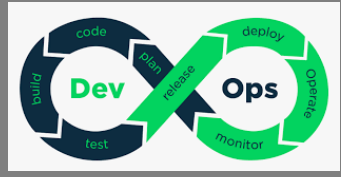
  - The desired outcome is that anyone with sufficient privileges to deploy a new release can do so at any time in one or a few clicks. By eliminating nearly all manual tasks, developers become more productive.
- Continuous Deployment
  - Every change in the source code is deployed to production automatically, without explicit approval from a developer.

  - As long as it passes the quality controls

NOTE:

a

CI/CD



What makes for good CI?

1. Decoupled stages
   - Each step in CI should do a single focused task
2. Repeatable
   - Automated in a way that is consistently repeatable
   - Tooling should work for local developers too – Local/Remote parity
3. Fail fast
   - Fail at the first sign of trouble

What makes for good CD?

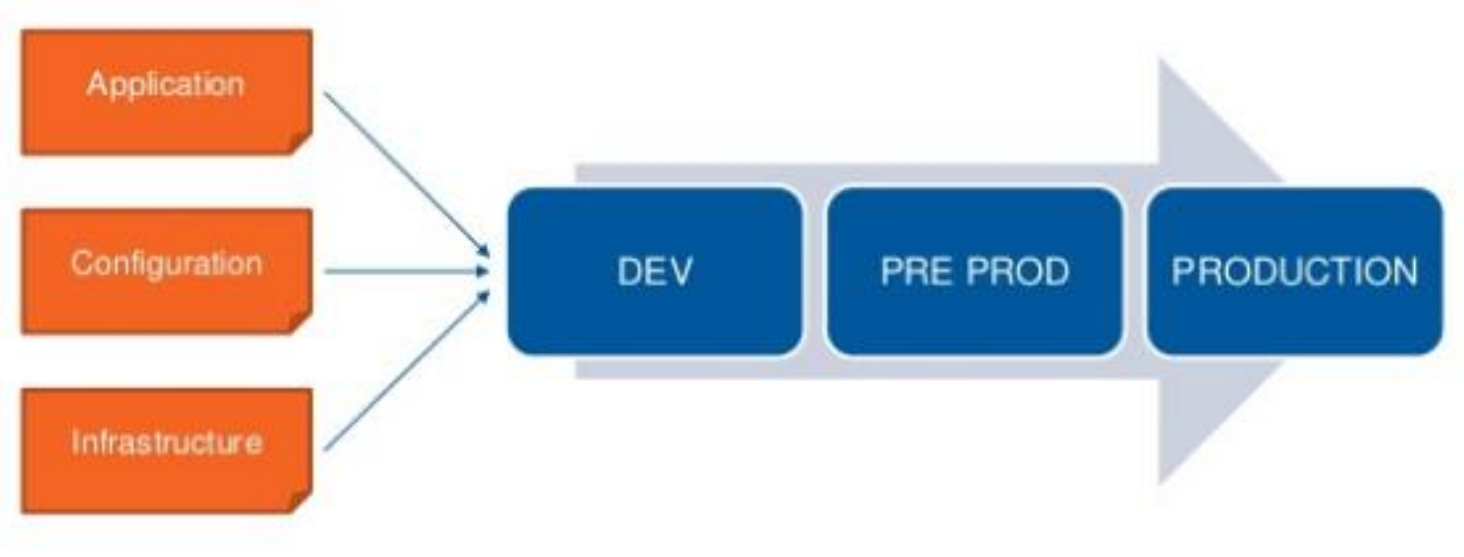
1. Design with the system in mind
   - Cover as many parts of a deployment as possible
   - Application | Infrastructure | Configuration | Data
2. Pipelines
   - Continually increase confidence as we move towards production
3. Globally unique versions
   - Know the state of the system at any time
   - Be able to demonstrate difference between current and future state

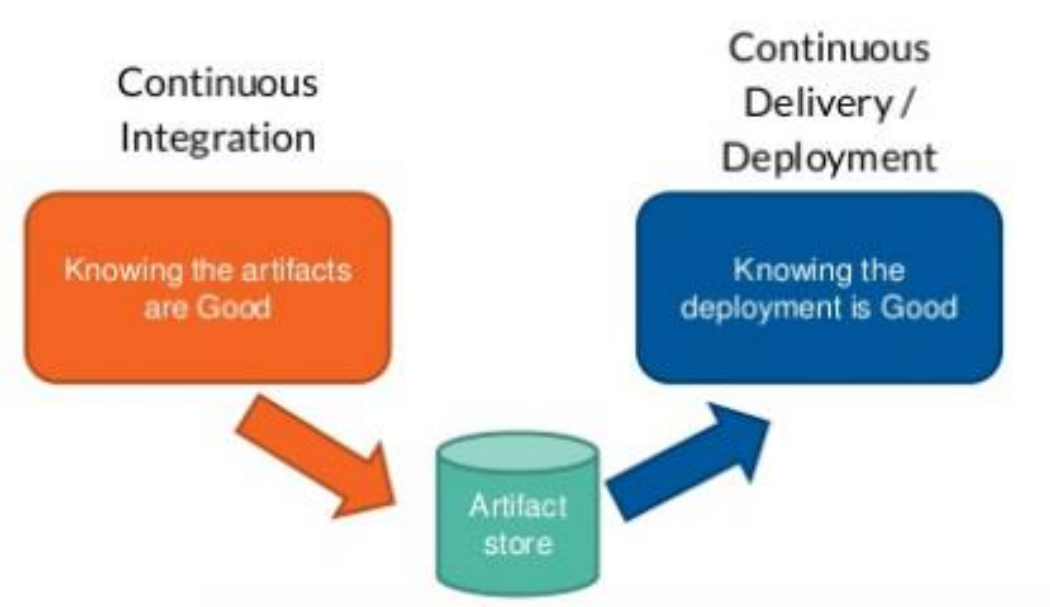
NOTE:

a

CI/CD

CD Pipeline and Integration with CI



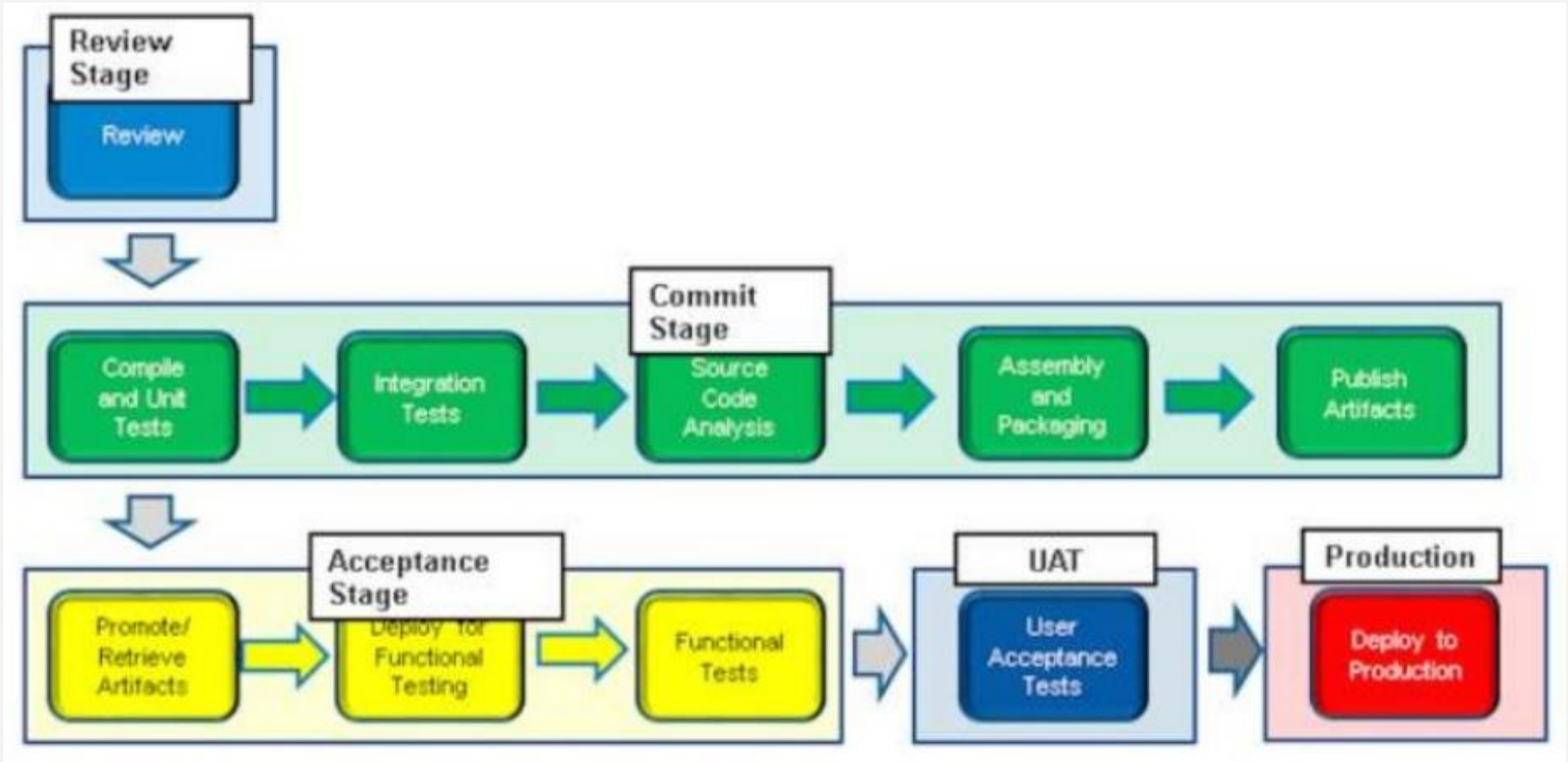


NOTE:

a

CI/CD

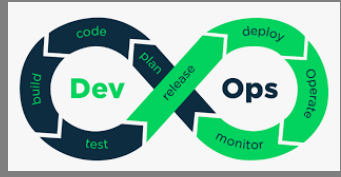
Full Deployment Pipeline



NOTE:

a

CI/CD



CI

- Version control system. This is the most basic and the most important requirement for implementing CI. ...
- Branching strategy. ...
- Self-triggered builds. ...
- Code coverage. ...
- Static code analysis. ...
- Automated testing. ...
- Binary repository tools. ...
- Automated packaging.

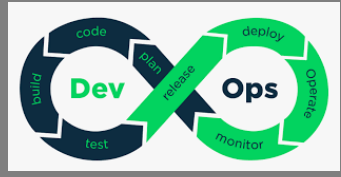
CD

- Sprints. Organizing work into short work cycles known as sprints that typically last a few weeks. ...
- Continuous Integration. ...
- Test Automation. ...
- Deployment Pipeline. ...
- User Acceptance Testing.

NOTE:

a

CI/CD



Tracking the progress

Unit testing and shifting left

- Make the tests easy to run
  - Run them locally
- Invest in good quality IDE support
  - Consistent entrypoint - Abstract the complexity
- Makefiles / Gradle
  - Pull request unit test execution

Failing fast

- Code inspection
  - Code linting
  - Security inspection
  - Code format
- Code coverage
  - Track coverage changes

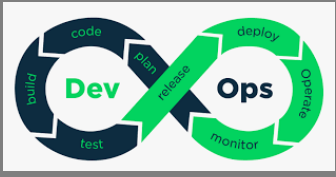
Centralised artifacts

- Managing output from a CI process
  - Only the strong survive
  - Central common location
  - Clearly understood name and version approach
  - Build them only once
  - Make them timeless – externalise configuration

NOTE:

a

CI/CD: Business benefits of CI/CD



- Reduction of delivery risk
  - No longer we need to rely on humans with specific knowledge as the gate-keepers of quality
  - Reduced chance of humans not following the process
  - Reduced chance of mis-communication on executing the change
- To encode the process, we need to know the process
  - If we know all the tests pass,
  - If we know all the steps in deployment,
  - What is stopping us from releasing?
- Better visibility on change
  - As our systems and tools are version controlled
  - And we know what the current state of production is
  - And we can describe the process by which it will be changed
  - We can diff the system states with confidence
- Opens up more avenues for review and increased audit compliance
- Increased efficiency and delivery options
  - Enables us to deliver things with reduced effort
  - This leads us to deploy change more frequently
  - Which leads to getting feedback faster
  - That enables us to experiment easier
  - This leads to smaller batch sizes
  - Which leads to and increased flow of the entire system
- Enhanced learning from failure
  - When we have an issue or failure, we write a test to cover it
  - This test gets added to our suite and executed every time
  - Decreases our risk of this issue occurring again

NOTE:

a

Software Delivery Management

a

10 CI/CD Best Practices Summary

1. Track work items
2. Use source code management
3. Tags, not Branches
4. Automate the builds
5. Stop the line when the build breaks
6. Validate and test
7. Deploy
8. Improve incrementally
9. Collaboration
10. Create a true DevOps culture

NOTE:

a

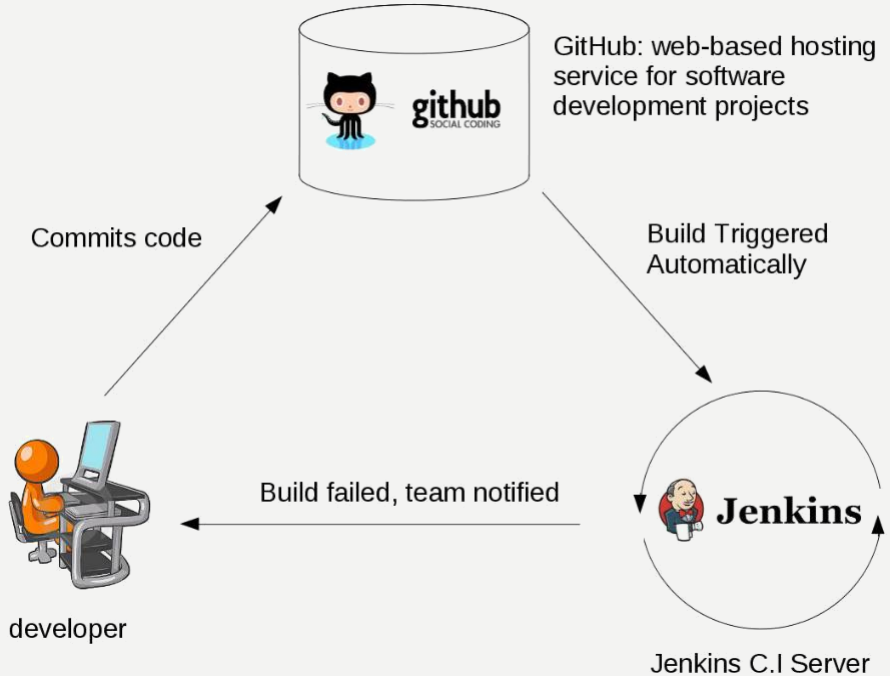
Jenkins

What is Continuous Integration?

- Developers commit code to a shared repository on a regular basis.
- Version control system is being monitored. When a commit is detected, a build will be triggered automatically.
- If the build is not green, developers will be notified immediately.

Why do we need Continuous Integration?

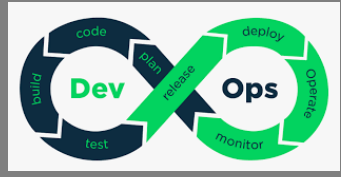
- Detect problems or bugs, as early as possible, in the development life cycle.
- Since the entire code base is integrated, built and tested constantly , the potential bugs and errors are caught earlier in the life cycle which results in better quality software.



NOTE:

a

Jenkins



Stage 1:

- No build servers.
- Developers commit on a regular basis.
- Changes are integrated and tested manually.
- Fewer releases.

Stage 2:

- Automated builds are scheduled on a regular basis.
- Build script compiles the application and runs a set of automated tests.
- Developers now commit their changes regularly.
- Build servers would alert the team members in case of build failure.

Stage 3:

- A build is triggered whenever new code is committed to the central repository.
- Broken builds are usually treated as a high priority issue and are fixed quickly.

Stage 4:

- Automated code quality and code coverage metrics are now run along with unit tests to continuously evaluate the code quality.

Stage 5:

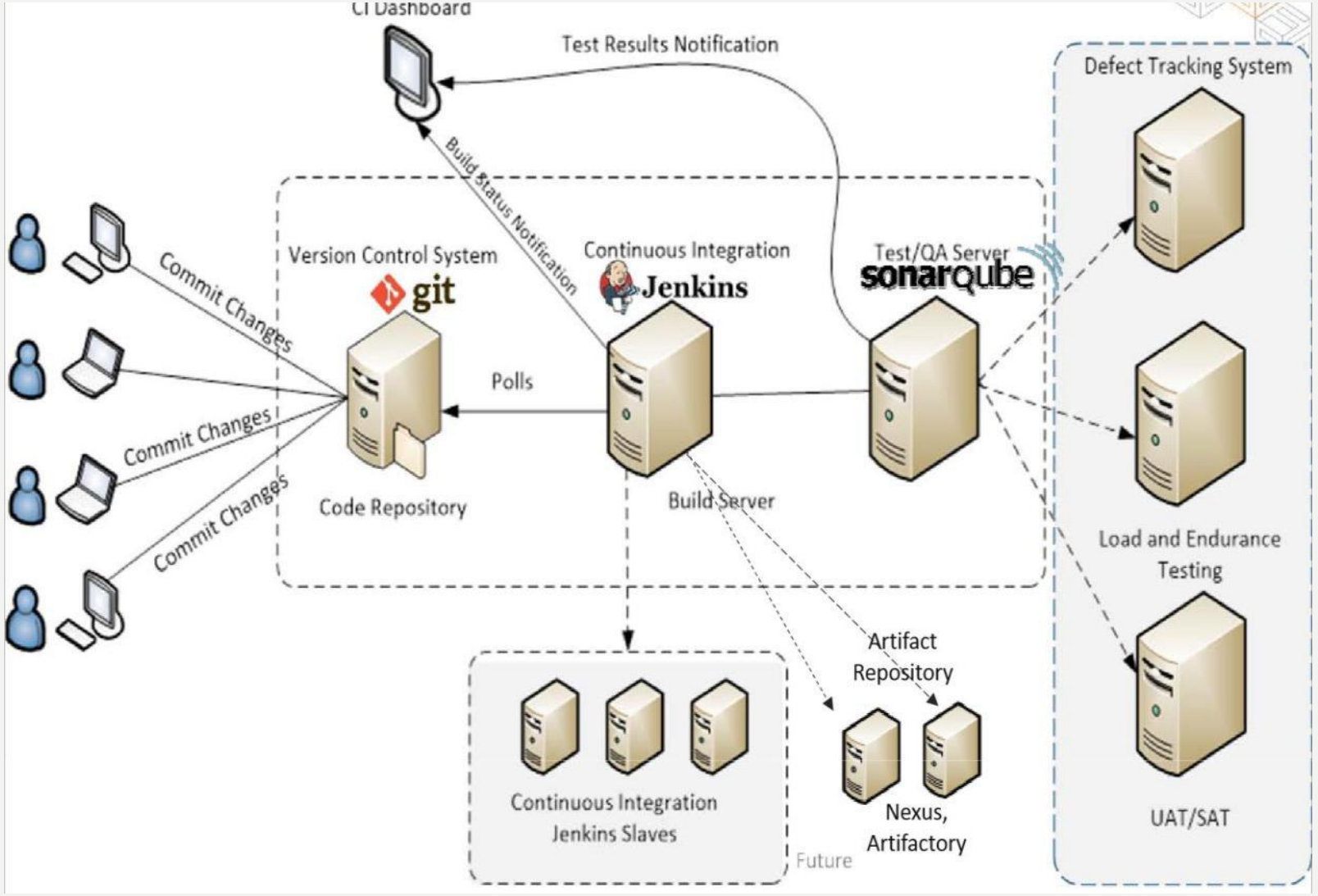
- Automated Deployment.

NOTE:

a

Jenkins

a

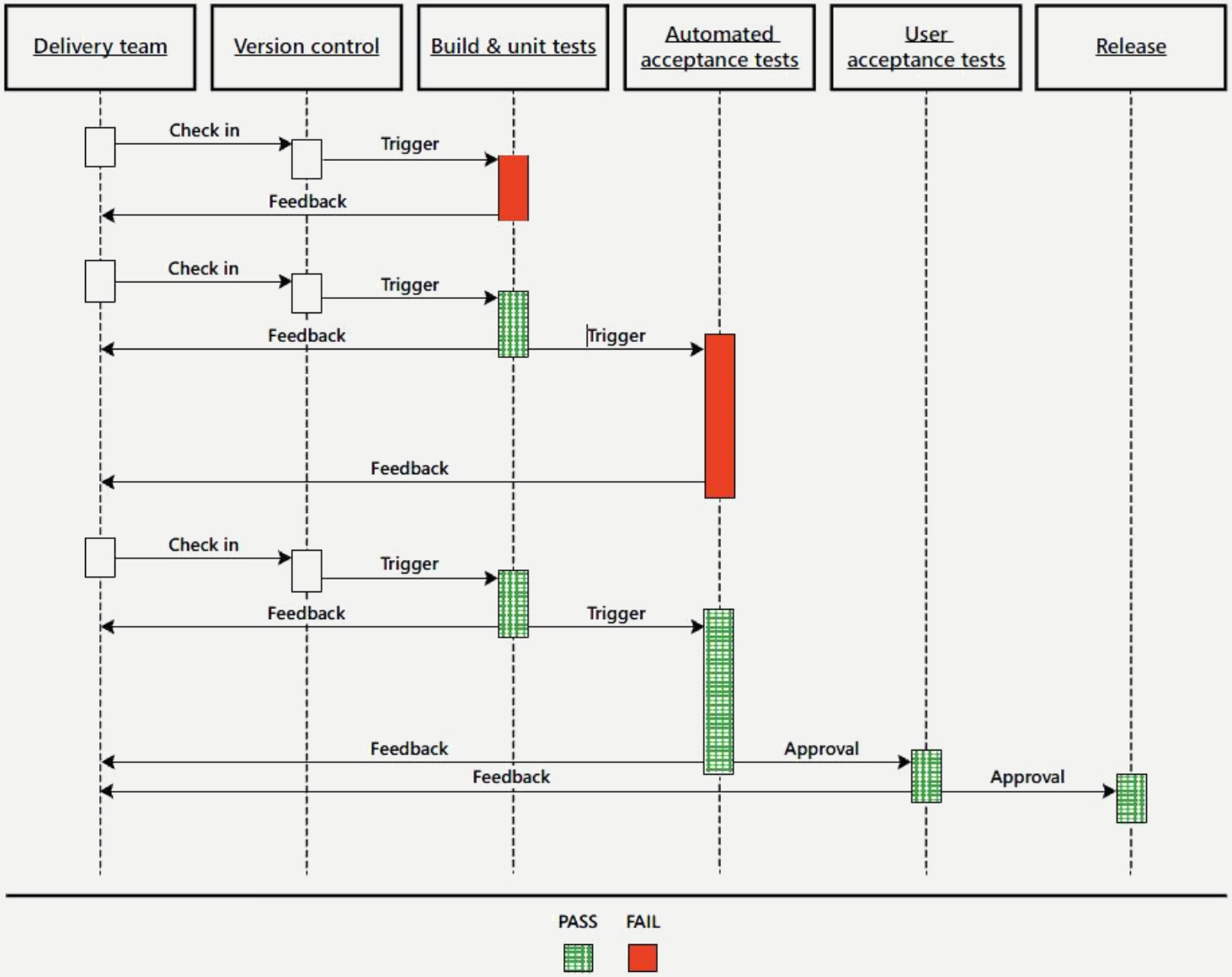


NOTE:

a

Jenkins

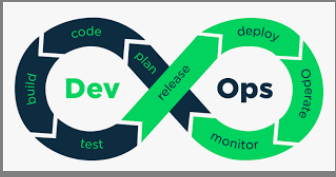
a



NOTE:

a

Jenkins



- Continuous Integration

The practice of merging development work with the main branch constantly.

- Continuous Delivery

Continual delivery of code to an environment once the code is ready to ship. This could be staging or production. The idea is the product is delivered to a user base, which can be QAs or customers for review and inspection.

- Continuous Deployment

The deployment or release of code to production as soon as it is ready.

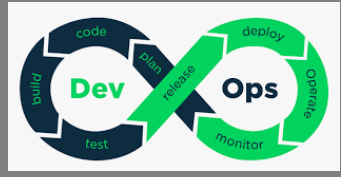
Continuous Integration is also a mindset

- Fixing broken builds should be treated as a high priority issue for all team members.
- The deployment process should be automated, with no manual steps involved.
- All team members should focus on contributing to high-quality tests because the confidentiality of the CI process highly depends on the quality of the tests.

NOTE:

a

Jenkins: How to Implement CI



What is Jenkins

- Jenkins is a continuous integration and build server.
- It is used to manually, periodically, or automatically build software development projects.
- It is an open source Continuous Integration tool written in Java.
- Jenkins is used by teams of all different sizes, for projects with various languages.

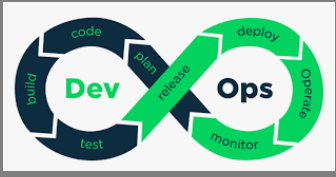
Why Jenkins is popular

- Easy to use
- Great extensibility
  - Support different version control systems
  - Code quality metrics
  - Build notifiers
  - UI customization

NOTE:

a

Jenkins: Master and Slave



Jenkins' Master and Slave Architecture

Master:

- Schedule build jobs.
- Dispatch builds to the slaves for the actual job execution.
- Monitor the slaves and record the build results.
- Can also execute build jobs directly.

Slave:

- Execute build jobs dispatched by the master.

NOTE:

a

Jenkins: Tools Integration

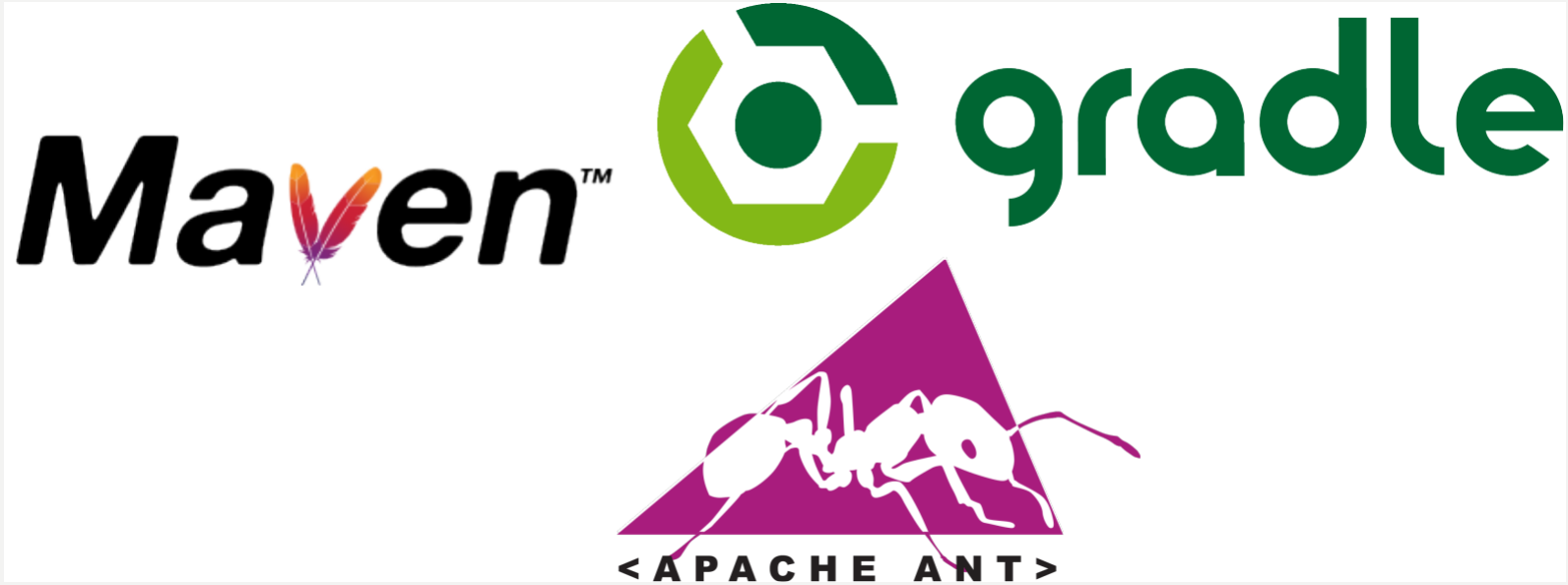
Install GIT and GitHub plugin
Install and Configure Maven

What does Maven do?

- Maven describes how the software is built.

- Maven describes the project's dependencies.

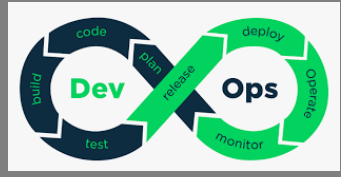
Java Build Tools



NOTE:

a

Jenkins: Tools Integration



Configure Jenkins for a Maven -based project
Create a Maven -based Jenkins project
Run Maven-based Jenkins project

Maven pom.xml file

- Describe the software project being built, including
  - The dependencies on other external modules.
  - The directory structures.
  - The required plugins.
  - The predefined targets for performing certain tasks such as compilation and packaging.

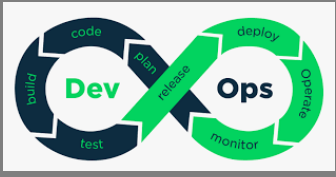
Different Phases in Maven Build Lifecycle

- validate Validate the project is correct and all necessary information is available.
- compile Compile the source code of the project.
- test Test the compiled source code using a suitable unit testing framework.
- package Take the compiled code and package it in its distributable format.
- verify Run any checks on results of integration tests to ensure quality criteria are met.
- install Install the package into the local repository, for use as a dependency in other projects locally.
- deploy Copy the final package to the remote repository for sharing with other developers and projects.

NOTE:

a

Jenkins: Tools Integration - Maven Build Phases



Maven Build Phases

- These lifecycle phases are executed sequentially to complete the default lifecycle.
- We want to specify the maven package command, this command would execute each default life cycle phase in order including validate, compile, test before executing package.
- We only need to call the last build phase to be executed.

Jenkins code quality metrics report

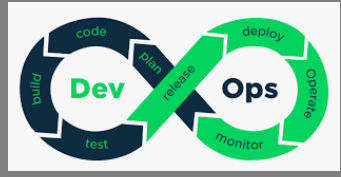
Checkstyle is a code static analysis tool to help programmers to write Java code that adheres to a coding standard such as

- Avoiding multiple blank lines;
- Removing unused variables;
- Enforcing correct indentations;

NOTE:

a

Jenkins: Tools Integration - Maven Build Phases



Jenkins' support for other build systems (Ant, Gradle and shell scripts)

Apache Ant

- Widely-used and very well-known build scripting language for Java.
- Flexible, extensible, relatively low-level scripting language.
- An Ant build script is made up of a number of targets, each target performs a particular job in the build process.

Gradle

- Gradle is a relatively new open source build tool for the Java Virtual Machine.
- Build scripts for Gradle are written in a Domain Specific Language based on Groovy.
- The concise nature of Groovy scripting lets you write very expressive build scripts with very little code.

Install and configure Tomcat as a staging environment

Tomcat

Tomcat is an open-source web server and provides a "pure Java" HTTP web server environment in which Java code can run.

- Install copy artifact and deploy to container plugins
- Deploy our application to staging environment

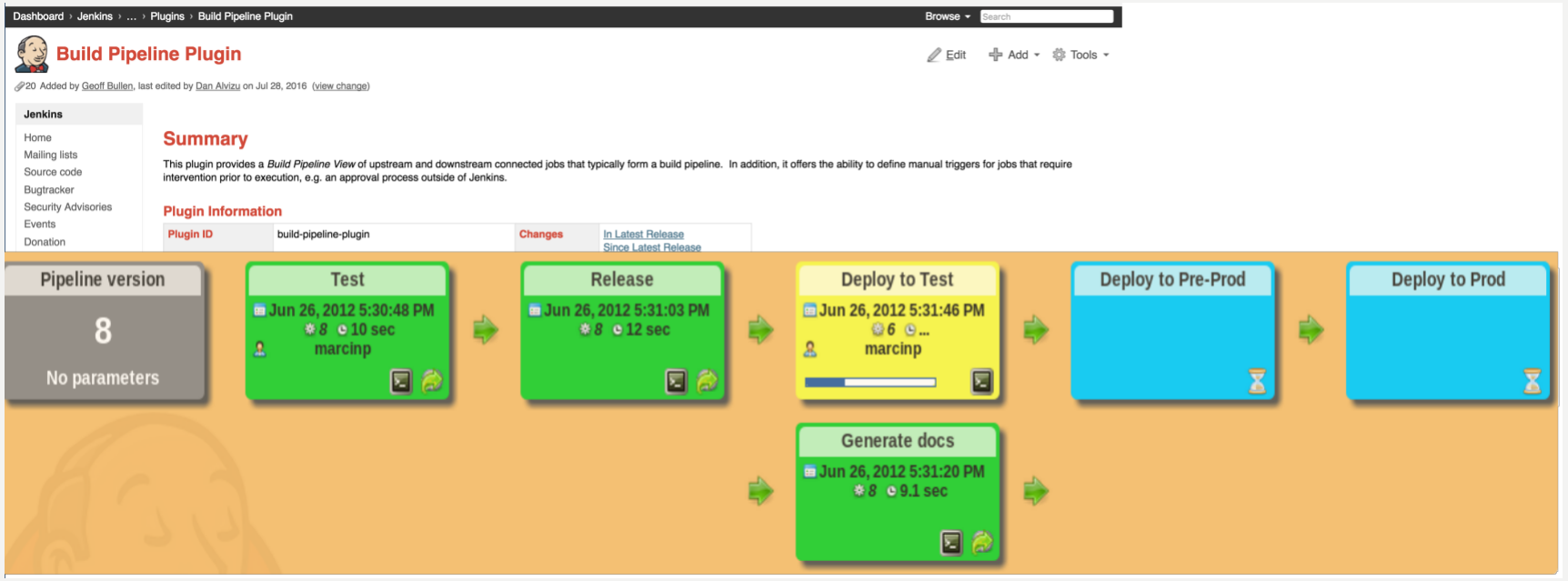
NOTE:

a

Jenkins: Jenkins Build Pipeline

Build Pipeline Plugin
Parallel Jenkins Build
Continuous Delivery
Deploy our app to production
Benefits of a code-based pipeline

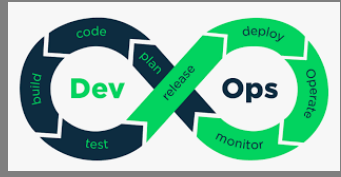
- Version control
- Best Practices
- Less error-prone execution of jobs
- Logic-based execution of steps



NOTE:

a

Jenkins: Parallel Jenkins Build



Parallel Jenkins Build
Continuous Delivery
Deploy our app to production

Benefits of a code-based pipeline

- Version control
- Best Practices
- Less error-prone execution of jobs
- Logic-based execution of steps

Additional automation

- Setup Git repository polling
- Deployment to our tomcat servers
- We will setup tasks to run in parallel

Steps

- Step 1: Configure securit groups for Tomcat servers and create key pairs.
- Step 2: Provision instances to staging and production environments.
- Step 3: Install and run Tomcat on created instances.
- Step 4: Fully automate our existing Jenkins pipeline.

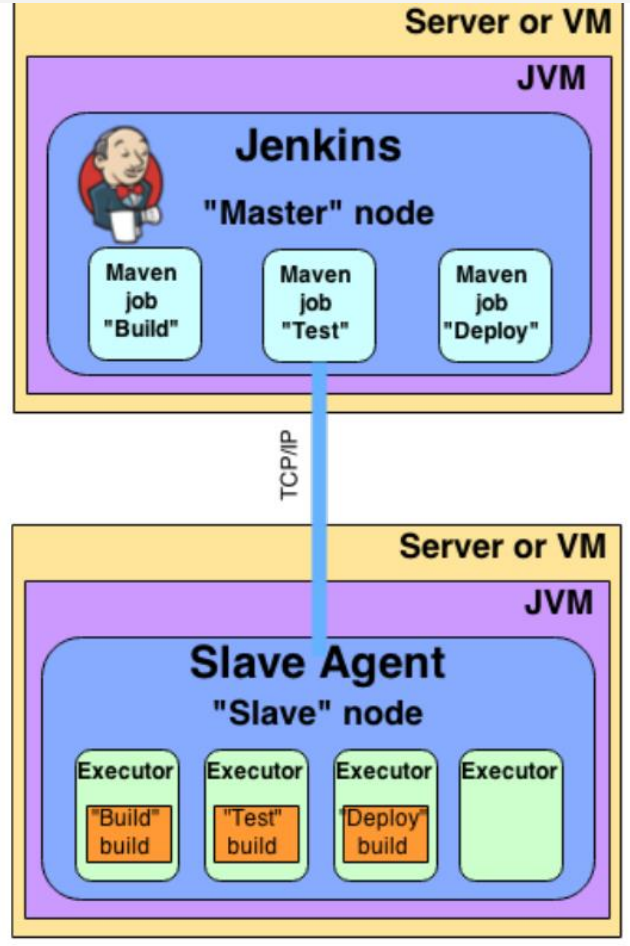
NOTE:

a

Jenkins: Distributed Jenkins Builds

Install Jenkins Master in the Cloud

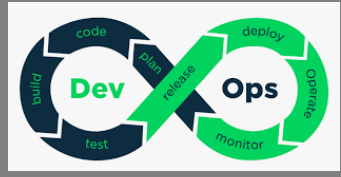
Jenkins Slave Agent



NOTE:

a

Jenkins: Distributed Jenkins Builds



Install Jenkins slaves in the cloud and form a Jenkins cluster
Concurrent Builds on Jenkins Cluster Label Jenkins Nodes

Build Orchestration: Jenkins

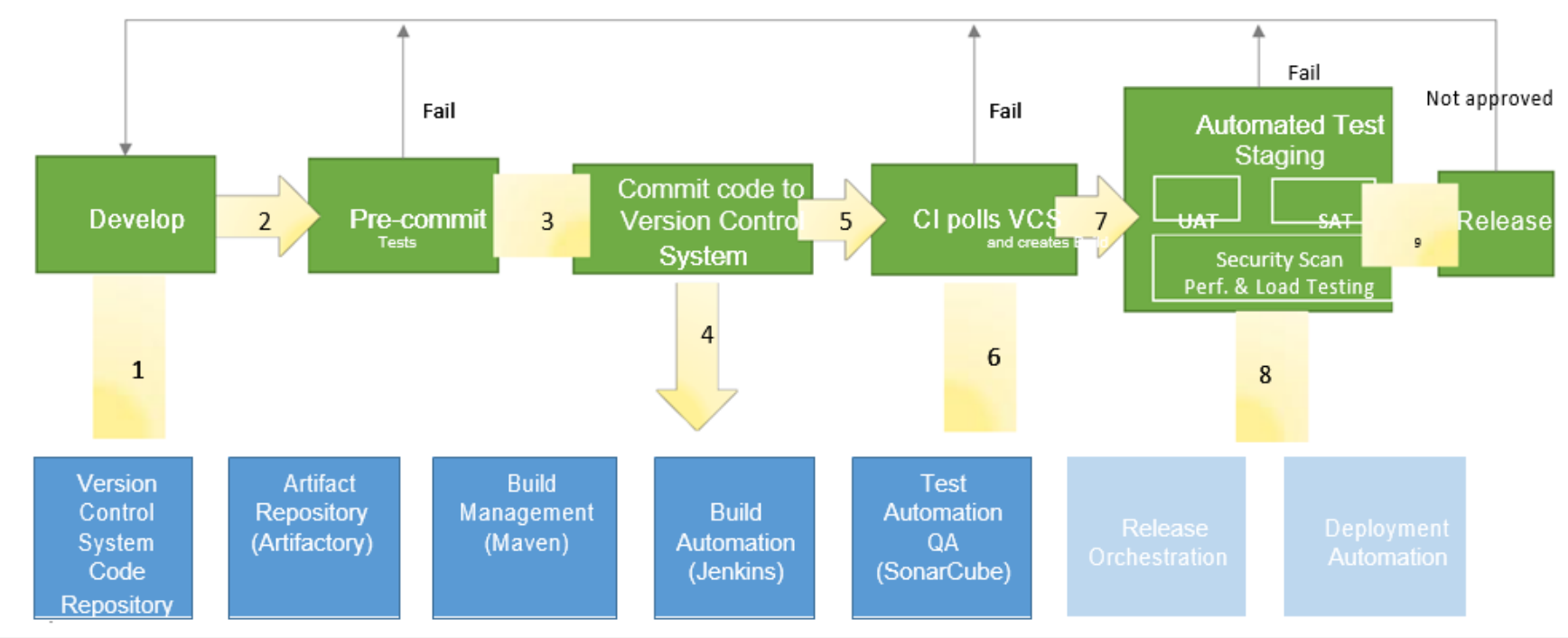
- Continuous integration system
- Enable automated build and test process
- Can monitoring executions of externally-run jobs, such as cron jobs and procmail jobs…
- Dependency tracking, allowing file finger printing and tracking for example which build is using which version of jars…
- Generates list of changes made to build from Subversion
- Distributed build/test
- Jenkins is a build orchestration, CI software
- building/testing software projects continuously
- monitoring executions of externally-run jobs
- FishEye allows you to extract information from your source code repository and display it in sophisticated reports.
- Crucible allows you to request, perform and manage code reviews.
- Subversion centralized version control system
- Sonar is a quality management platform for analyzing and measuring source code quality.

NOTE:

a

Jenkins: Functional Architecture

a



NOTE:

a