


- 
- Software Configuration Management
 - Version Control System
 - Basic Workflow

SOFTWARE CONFIGURATION MANAGEMENT

Software Configuration Management(SCM) is a process to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle.

The primary goal is to increase productivity with minimal mistakes.

SCM is part of cross-disciplinary field of configuration management and it can accurately determine who made which revision.

WHY DO WE NEED SOFTWARE CONFIGURATION MANAGEMENT?

- There are multiple people working on software which is continually updating.
- It may be a case where multiple version, branches, authors are involved in a software config project, and the team is geographically distributed and works concurrently.
- Changes in user requirement, policy, budget, schedule need to be accommodated.
- Software should be able to run on various machines and Operating Systems.
- Helps to develop coordination among stakeholders.
- SCM process is also beneficial to control the costs involved in making changes to a system.



VERSION CONTROLLING



WHAT IS VERSION CONTROL SYSTEM?

Version control systems are software tools that help software teams to manage changes to **source** code over time.

Version control software keeps track of every modification to the code in a special kind of database.

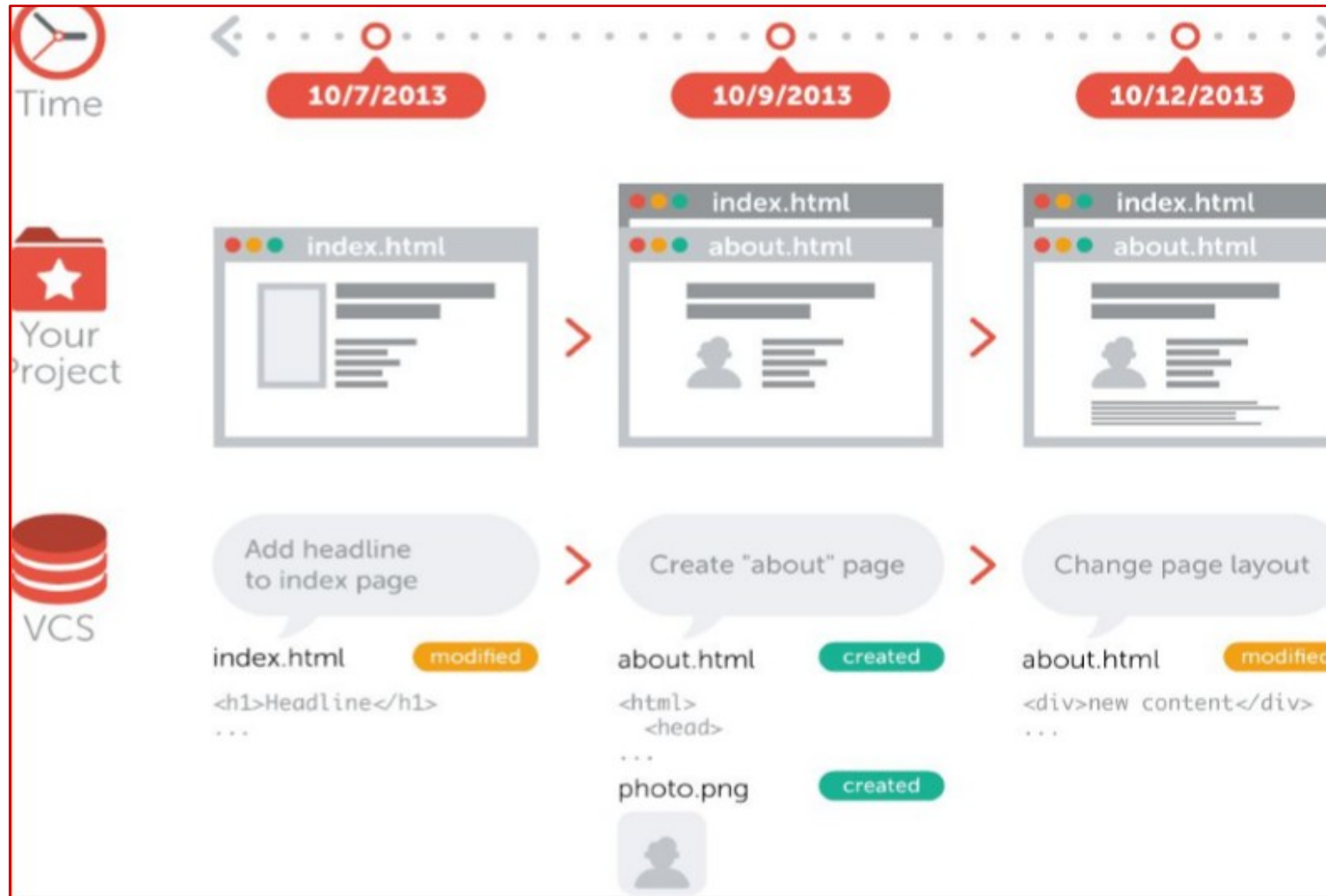
WHAT IS VERSION CONTROL SYSTEM?

You can think of a VCS as a kind of “database”.

It lets you save a snapshot of your complete project at any time you want.

When you later take a look at an older snapshot your VCS shows you exactly how it differed from the previous one.

WHAT IS VERSION CONTROL SYSTEM?



WHAT IS VERSION CONTROL SYSTEM?

Version control is independent of the kind of project / technology / framework you're working with.

It works just as well for an HTML website as it does for a design project or an iPhone app.

It lets you work with any tool you like; it doesn't care what kind of text editor, graphics program, file manager or other tool you use.



WHY TO USE VERSION CONTROL SYSTEM?



WHY TO USE VERSION CONTROL SYSTEM?

1- Collaboration

With a VCS, everybody on the team is able to work absolutely freely
- on any file at any time.

The VCS will later allow you to merge all the changes into a common version.

WHY TO USE VERSION CONTROL SYSTEM?

2- Storing Versions

Saving a version of your project after making changes is an essential habit. But without a VCS, this becomes tedious and confusing very quickly.

When you need it, you can request any version at any time and you'll have a snapshot of the complete project right at hand.

WHY TO USE VERSION CONTROL SYSTEM?

3- Restoring Previous Versions

Being able to restore older versions of a file or whole project.

If the changes you've made lately prove to be garbage, you can simply undo them in a few clicks.



TYPES OF VERSION CONTROL SYSTEM



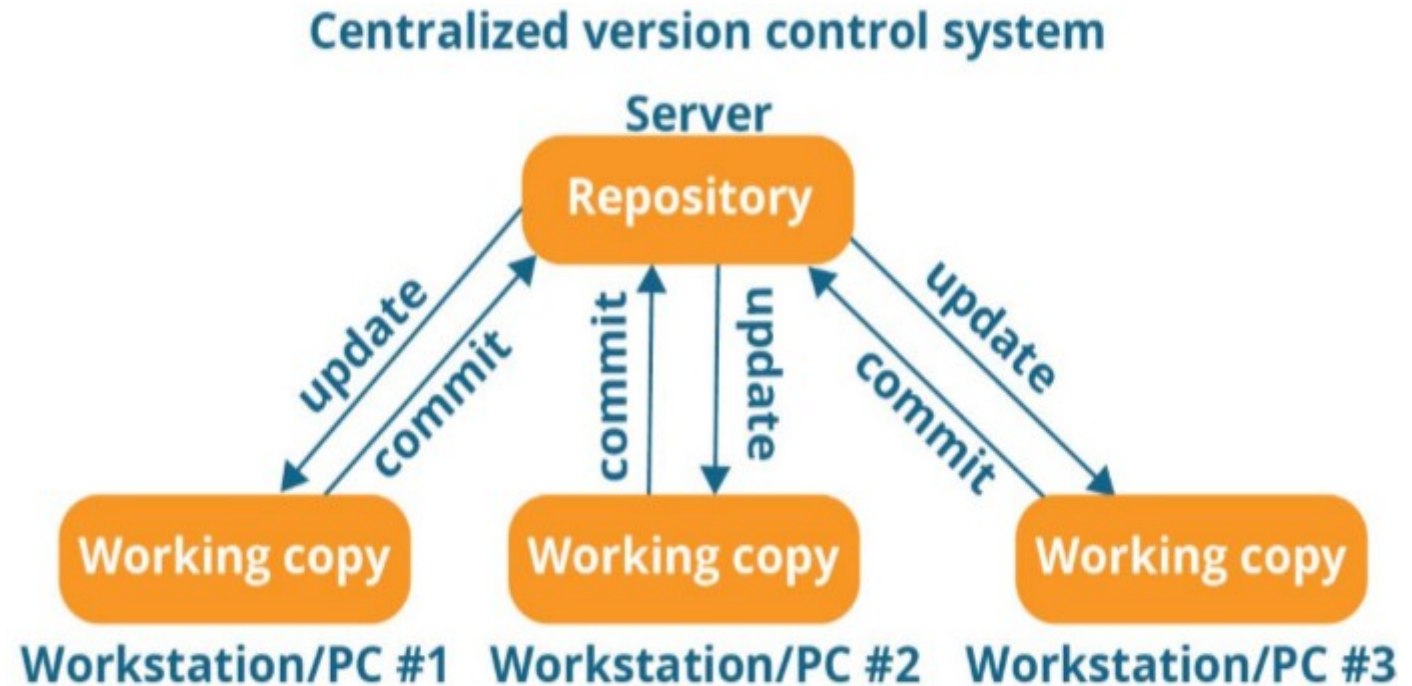
TYPES OF VERSION CONTROL SYSTEM

- Centralized Version Control System (CVCS)
- Distributed Version Control System (DVCS)

CENTRALIZED VERSION CONTROL SYSTEM

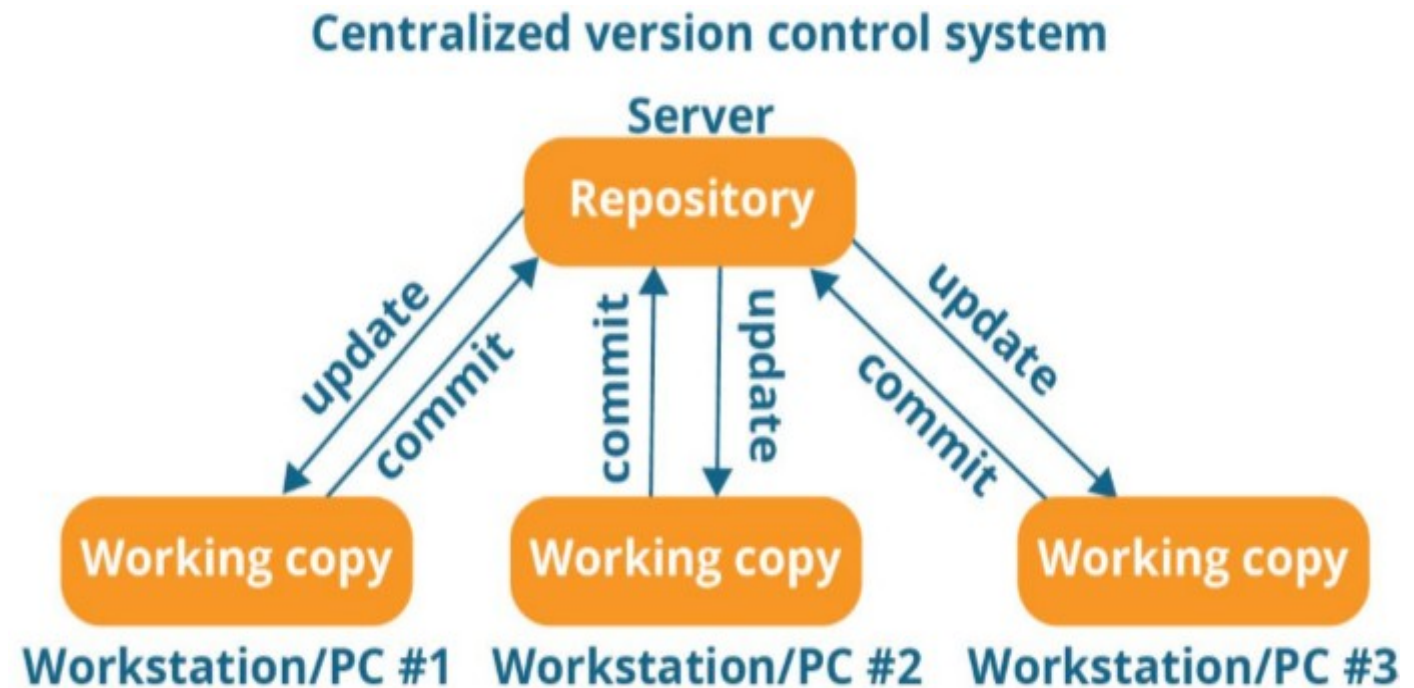
Centralized version control system (CVCS) uses a central server to store all files and enables team collaboration.

It works on a single repository to which users can directly access a central server.



DRAWBACKS OF CENTRALIZED VCS

- It is not locally available; meaning you always need to be connected to a network to perform any action.
- Since everything is centralized, in any case of the central server getting crashed or corrupted will result in losing the entire data of the project.





DISTRIBUTED VERSION CONTROL SYSTEM



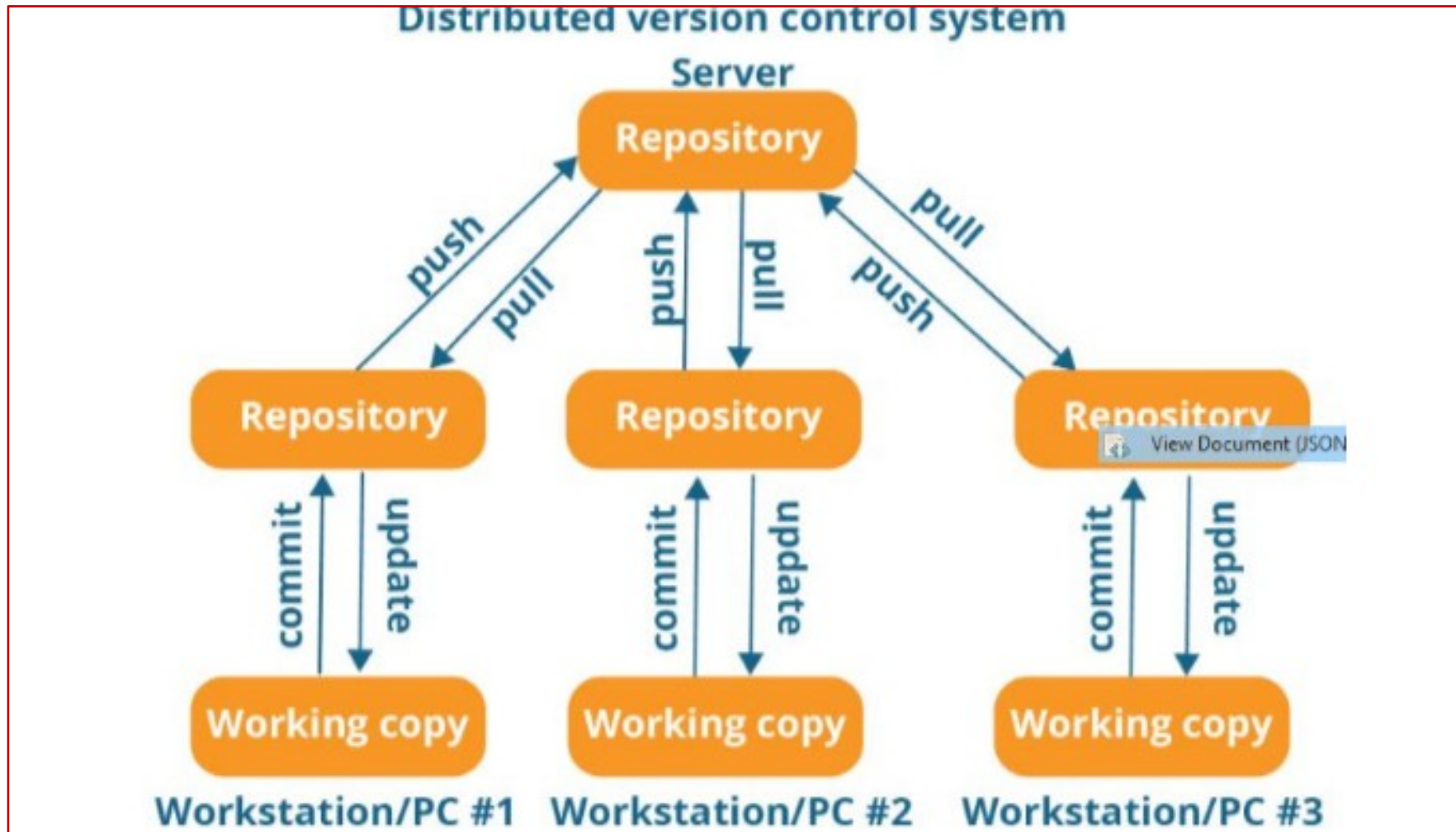
DISTRIBUTED VERSION CONTROL SYSTEM

These systems do not necessarily rely on a central server to store all the versions of a project file.

In Distributed VCS, every contributor has a local copy or “clone” of the main repository.

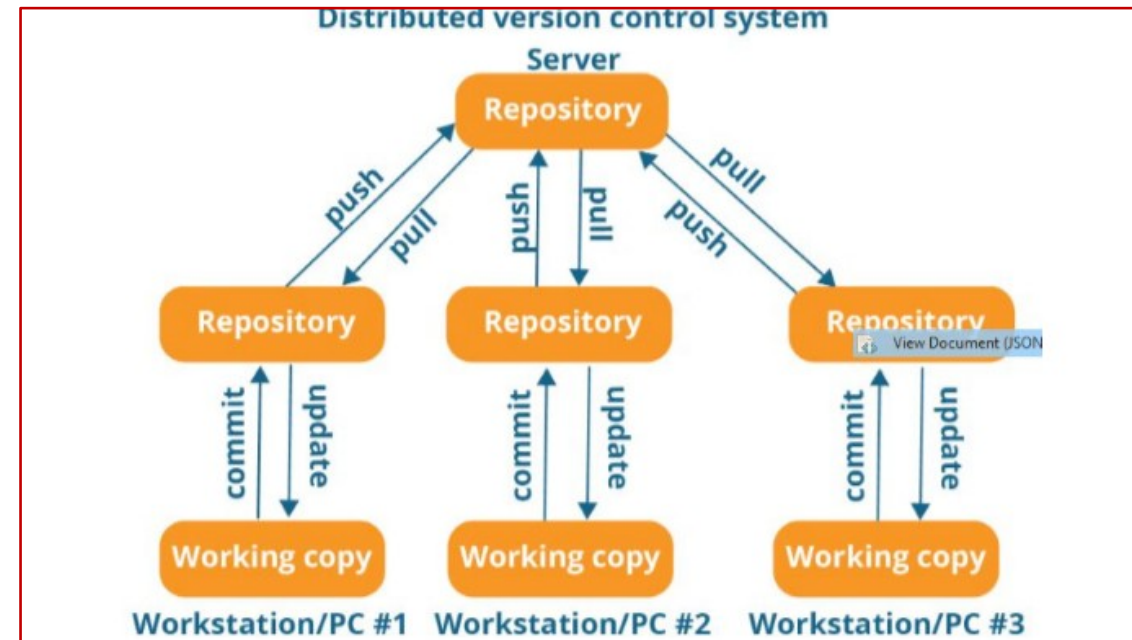
Everyone maintains a local repository of their own which contains all the files and metadata present in the main repository.

DISTRIBUTED VERSION CONTROL SYSTEM



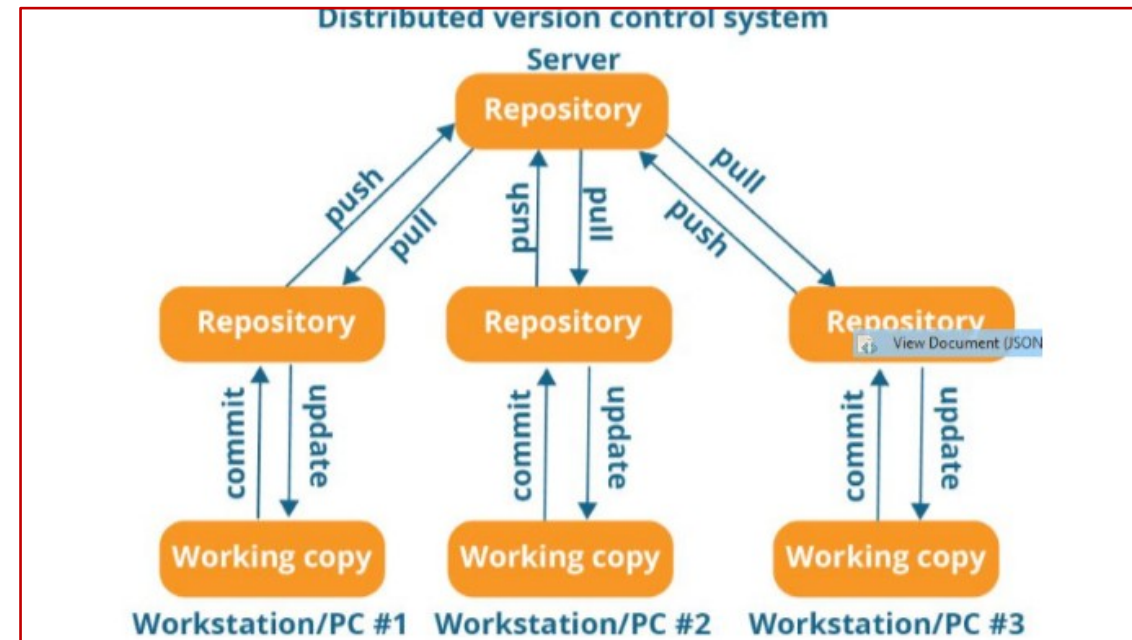
DISTRIBUTED VERSION CONTROL SYSTEM

- Every programmer maintains a local repository on its own, which is actually the copy or clone of the central repository on their hard drive.
- They can commit and update their local repository without any interference.



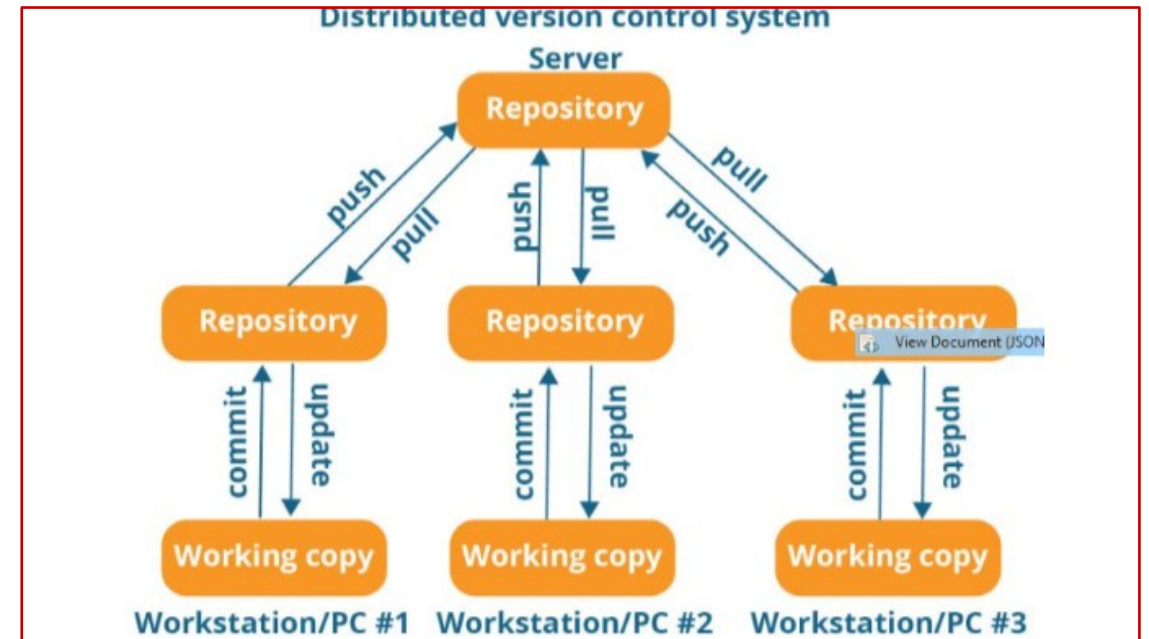
DISTRIBUTED VERSION CONTROL SYSTEM

- Programmer can update their local repositories with new data from the central server by an operation called “pull” and affect changes to the main repository by an operation called “push” from their local repository.



DISTRIBUTED VERSION CONTROL SYSTEM

- All operations (except push & pull) are very fast because the tool only needs to access the hard drive, not a remote server.
- Hence, you do not always need an internet connection.



ADVANTAGES OF DISTRIBUTED VCS

- Since every contributor has a full copy of the project repository, they can share changes with one another if they want to get some feedback before affecting changes in the main repository.
- If the central server gets crashed at any point of time, the lost data can be easily recovered from any one of the contributor's local repositories.

EXAMPLES OF VERSION CONTROL SYSTEM

- VSS (Microsoft Visual Source Safe)
- SVN (Apache Supervision)
- GIT

GIT

Git is a distributed version-control system for tracking changes in source code during software development.

INSTALLING AND SETTING UP GIT

<https://git-scm.com/downloads>

Smart GiT

<https://www.syntevo.com/smartgit/download>

INSTALLING AND SETTING UP GIT

Open Terminal and run following commands to setup your name and email id

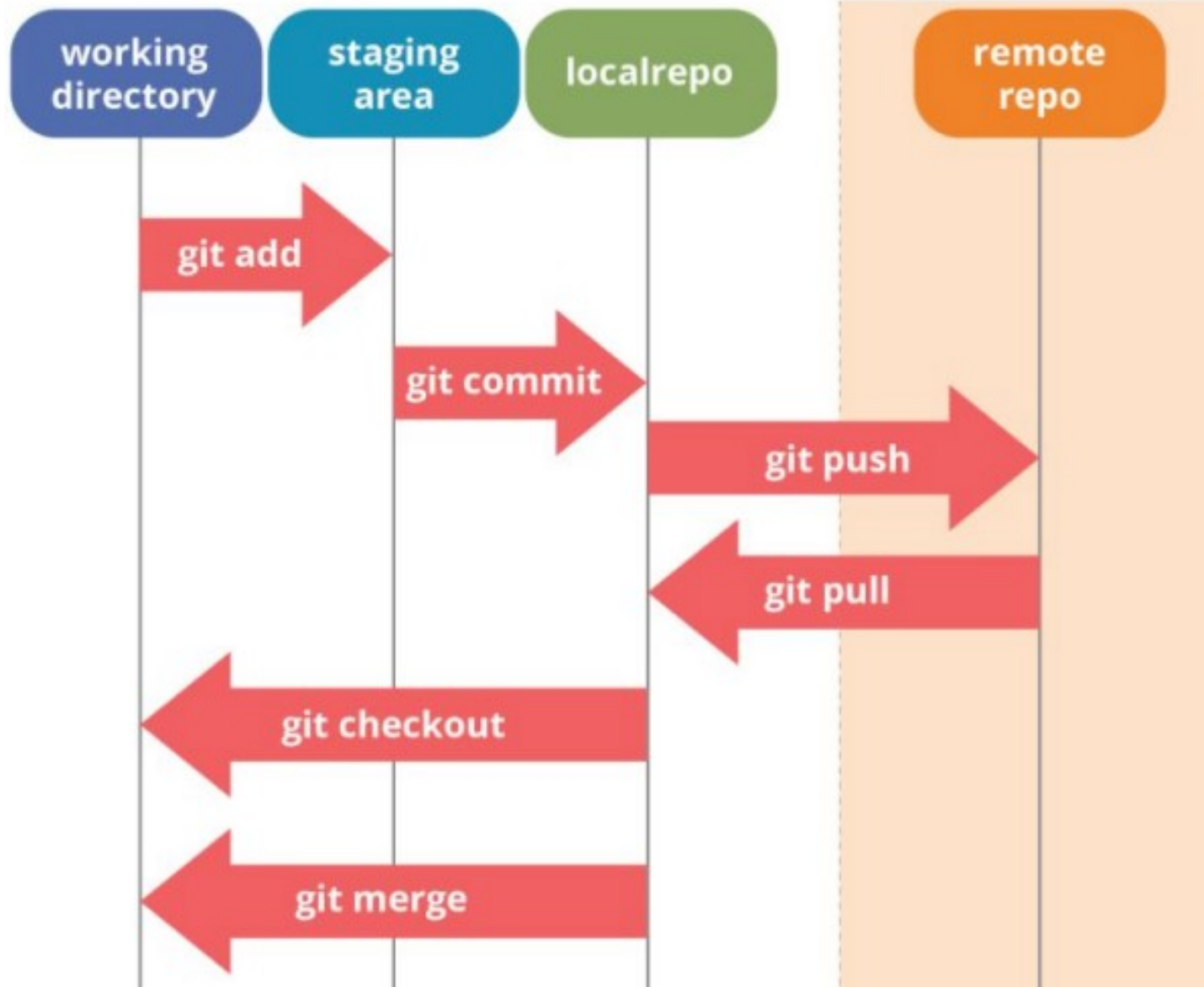
- `git config --global user.name "Your Name"`
- `git config --global user.email "Your Email"`

OPERATIONS IN GIT

- Initialize
- Add
- Commit
- Pull
- Push
- Branching
- Merging
- Rebasing
- Forking

Local

Remote



1. REPOSITORY

Think of a repository as a kind of database where your VCS stores all the versions and metadata.

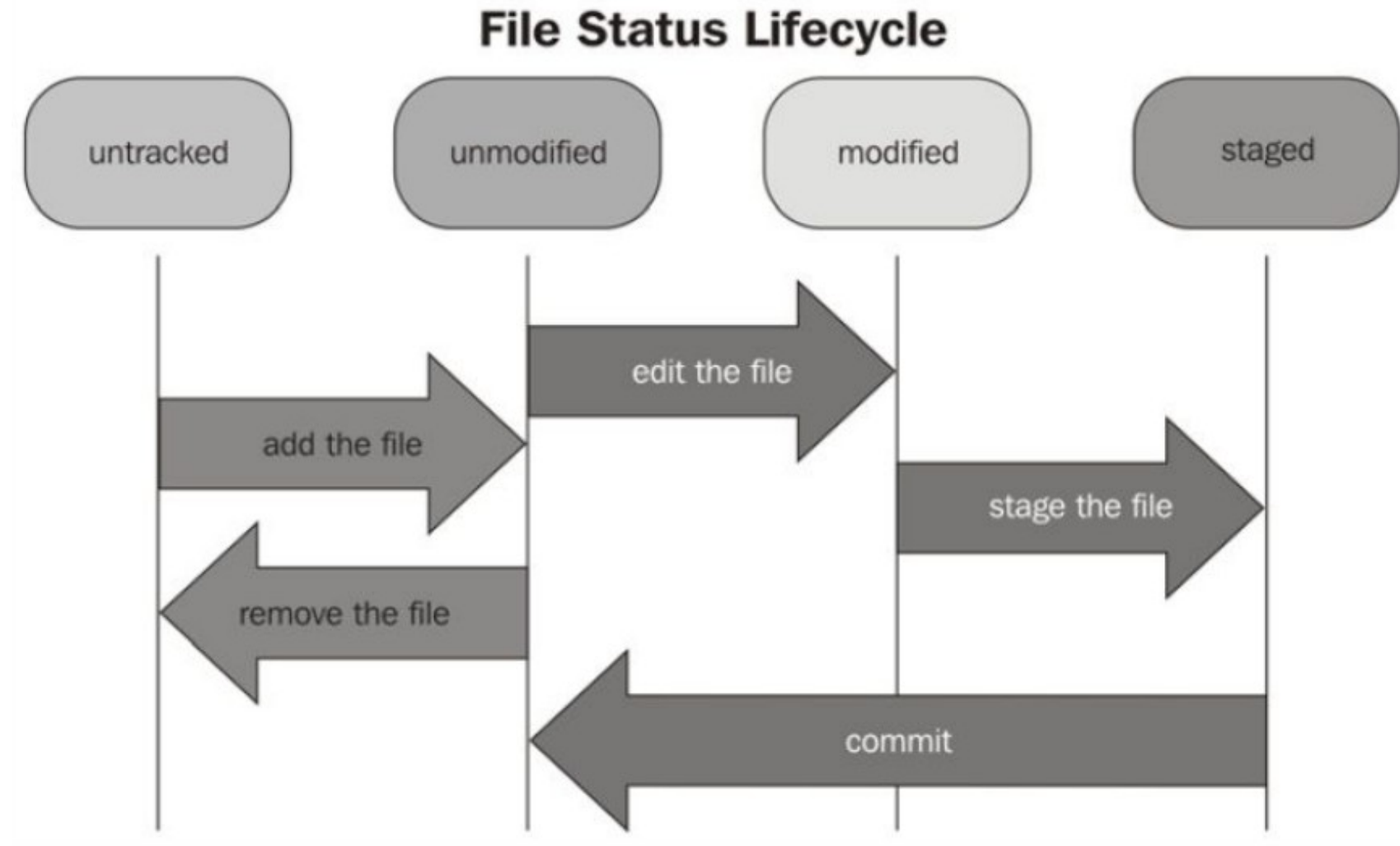
In Git, the repository is just a simple hidden folder named “.git” in the root directory of your project.

2. WORKING DIRECTORY

The root folder of your project is often called the “working copy” (or “working directory”).

It’s the directory on your local computer that contains your project’s files.

3. FILE STATUS



4. STAGING AREA

Staging area is a virtual place that collects all the files you want to include in the next commit.

In Git, simply making some changes doesn't mean they're automatically committed.

Every commit is “hand-crafted”: each change that you want to include in the next commit has to be marked explicitly (“added to the Staging Area” or, simply put, “staged”)

Working Copy

Your Project's Files



Git watches tracked files
for new local modifications...

Staging Area

Changes included in
the Next Commit

Local Repository

The ".git" Folder

Tracked (and modified)



If a file was modified since it
was last committed, you can
stage & commit these changes

stage



Changes that were added to
the Staging Area will be
included in the next commit

commit



All changes contained in a
commit are saved in the local
repository as a new revision



Changes that are **not staged** will
not be committed & remain as
local changes until you stage &
commit or discard them

Untracked



Changes in untracked files aren't
watched. If you want them included
in version control, you have to tell
Git to start tracking them. If not, you
should consider ignoring them.



BASIC WORKFLOW



BASIC WORKFLOW

1. Open terminal and create directory on your machine
2. Go into directory in terminal
3. Initialize repository in this directory
 - a. `git init`
 - b. This will create `.git` hidden folder in your directory which will make your current folder, a git repository

BASIC WORKFLOW

4. Create two files
5. Check status, it will show you two untracked files
 - a. `git status`
6. Add these files to staging area
 - a. Three ways to add files in staging
 - i. `git add filename-1 filename-2` OR
 - ii. `git add .`
 - iii. `git add *.html`

BASIC COMMANDS

- `git status`
- `git commit -m "commit message"`
- `git log`

UNSTAGE FILES OR REMOVE CHANGE

- `git reset` command will remove file from staging area.
- `git reset` can remove changes in files if they are not committed by using `git reset --hard` command.

IGNORING FILES

- There are a couple of files in a project that you don't want to be version controlled so.
 - Create an empty file in your favorite editor and save it as ".gitignore" in your project's root folder.
 - You can define rules in ".gitignore" file to ignore files.
 - Add file or directory path or extension or name.

IGNORING FILES

Ignore one specific file

- `path/to/file.ext`

Ignore all files with a certain name (anywhere in the project)

- `filename.ext`

Ignore all files of a certain type (anywhere in the project)

- `*.ext`

Ignore all files in a certain folder:

- `path/to/folder/*`



BASIC WORKFLOW DEMO WITH TERMINAL



BASIC WORKFLOW DEMO WITH SMART GIT



HAVE A GOOD DAY!