GIT

Pamela Solano

Goals

This Note should be give you the *introduction* to perform basic *distributed revision control* for *your own/team projects*.

References

(Chacon and Straub 2014)

Target

- No experience in *Version Control*
- Version control system but not *GIT*
- *GIT GUI*'s users (visual tools) not *command-line*.

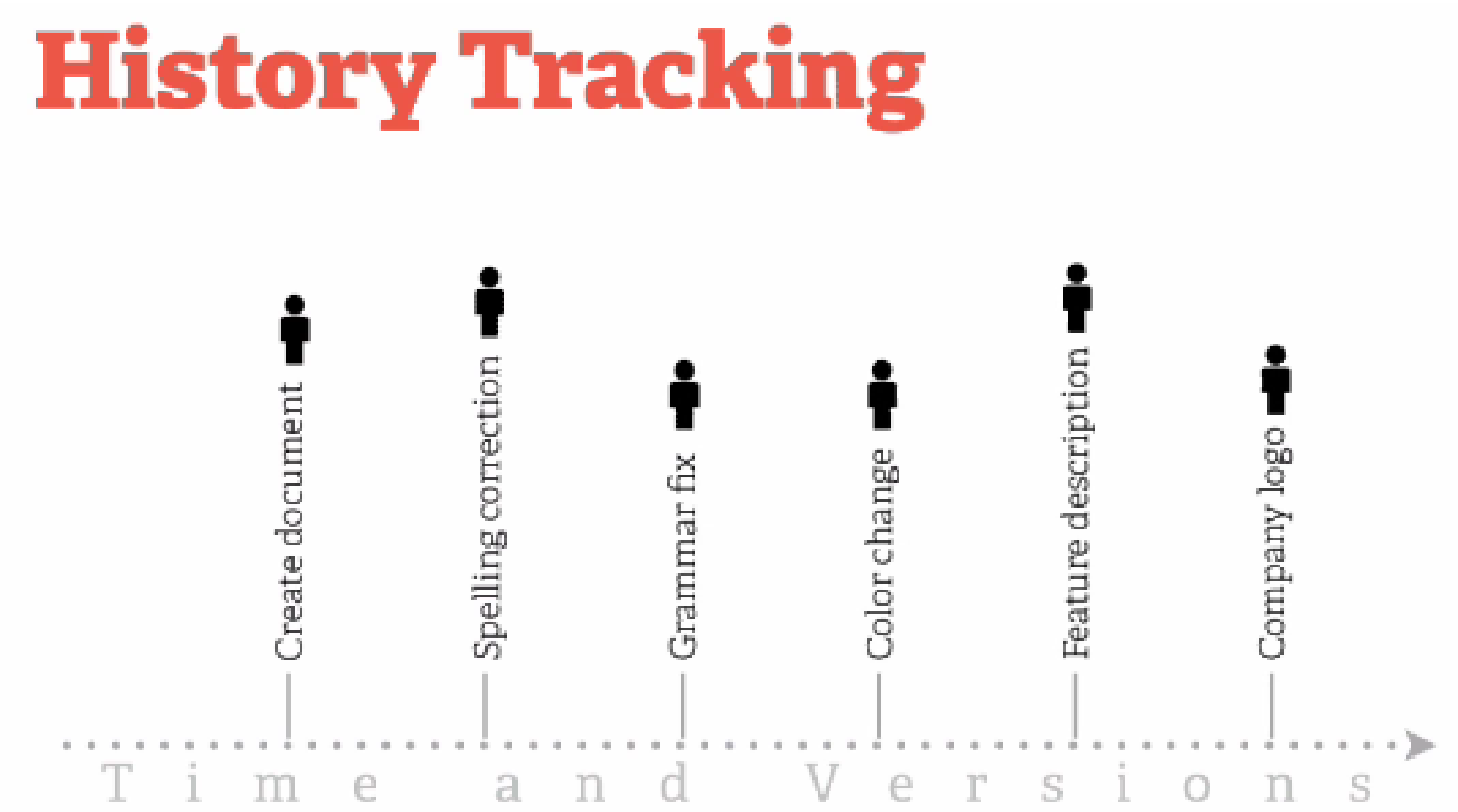
Your job and daily task

- Software developer
- Designer
- someone who write code

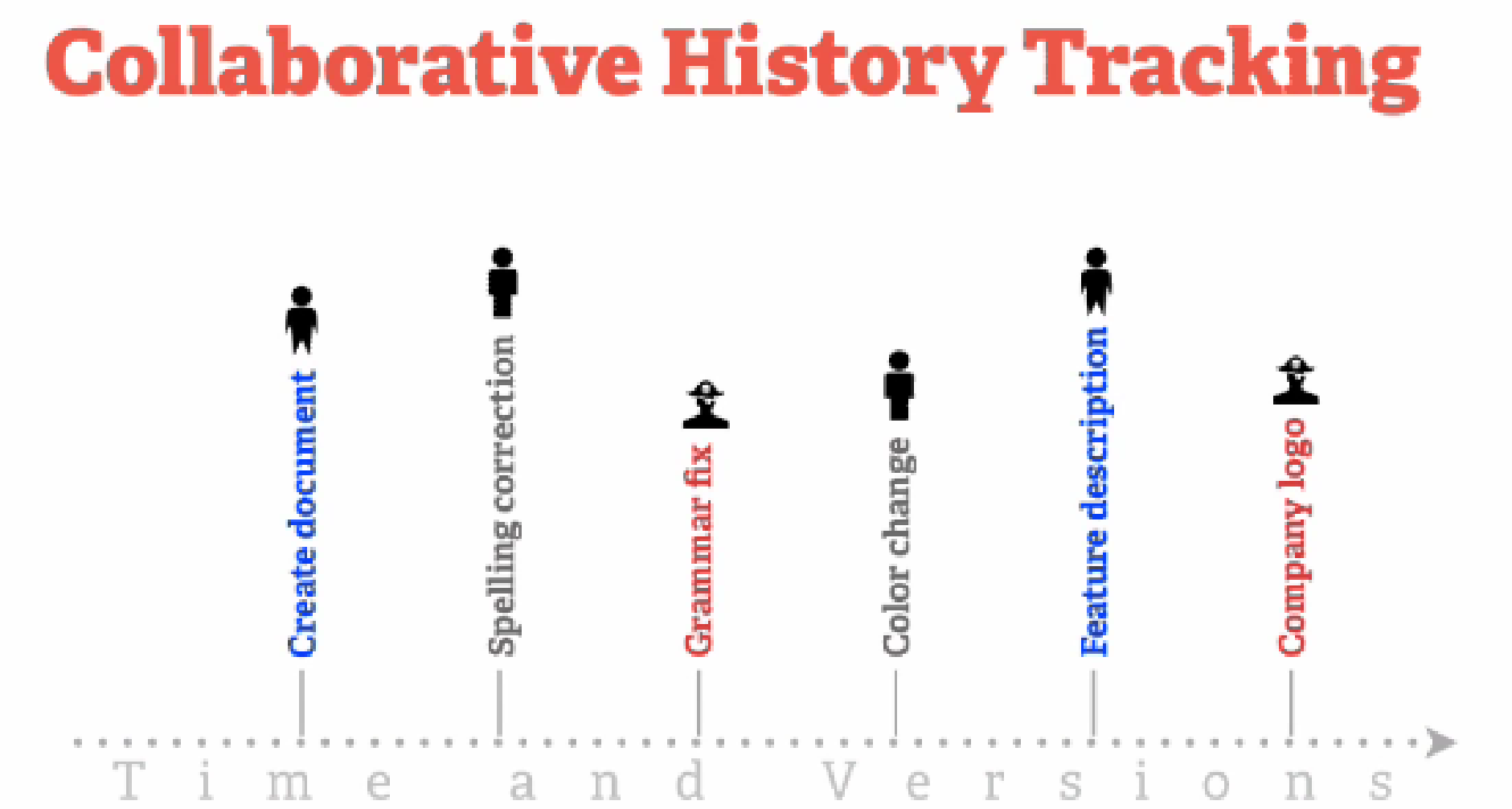
your task

- *Create* things
- *Save* things
- *Edit* things (correction-request of modification)
- Save the thing *again* and *again*

Personal changes

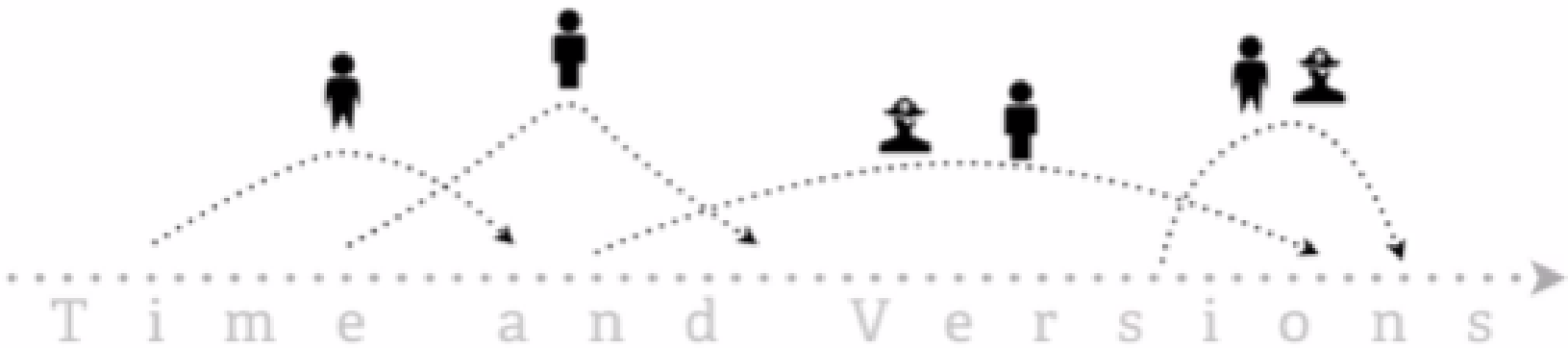


Team work


Collaborative History Tracking
Create document
Spelling correction
Grammar fix
Color change
Feature description
Company logo
Time and Versions

Non-Sequential Overlaping

Collaborative History Tracking



Track modifications: When-Why-what-who-merge

! Important

is the goals *VERSION CONTROL*

Why GIT? 1

- GIT is a fast and modern
  - implementation of *version control*
- GIT provides a *history* of content changes
- individual bases, tracking each piece file's content (graphics design, programs and code)

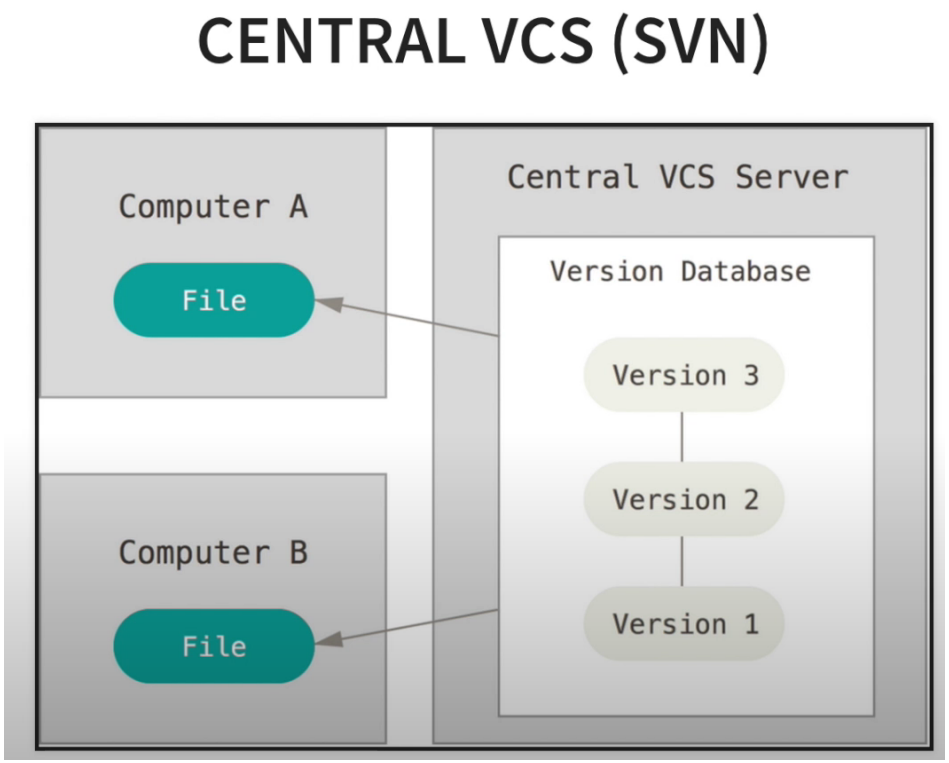
Why GIT? 2

- GIT facilitates *collaborative changes* to files.
- Not just a one person bringing modification and for all the team in the same time.
- people simultaneity change, working at the same time about an idea.
- GIT is easy to use for -any type of knowledge worker. -any time

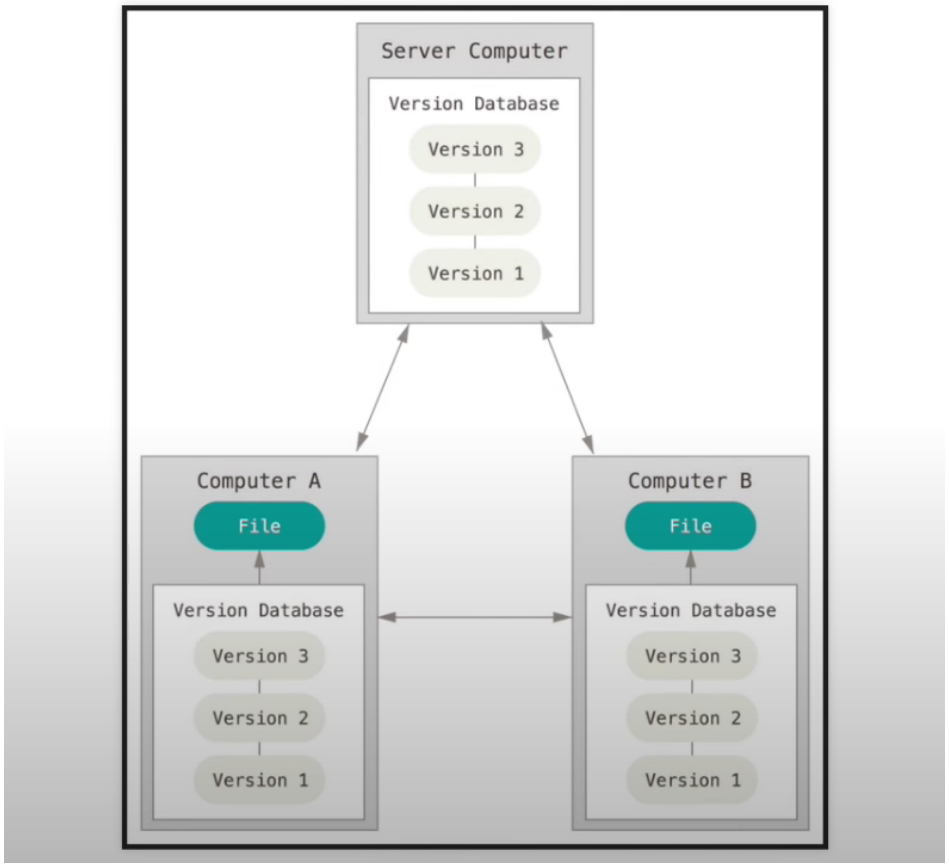
Types of Version Control?

What is the difference between *CENTRAL* and *DISTRIBUTED* Version Control System.

Without GIT


CENTRAL VCS (SVN)
Computer A
File
Computer B
File
Central VCS Server
Version Database
Version 3
Version 2
Version 1

DISTRIBUTED VCS (GIT)



GIT

Version control

- Local GIT
  - it is able just with a piece of code, it is locally neighborly just lines, just simple piece of commands.
- *Distributed* so that connectivity *doesn't* block
  - You don't need every complicated software to collaborate with your colleges
- *Easy* so that learning its commands can happen *progressively*

Summary

Important

with GIT you have a way **how track** your/team project progress

Tip

To fully understand the depth and power of Git you need to understand two simple ideas (LOCAL-REMOTE) on which it is based.

Get going with git

- How install
- setup
- configure
- and make your first use of the command line

Installing GIT

Official GIT's homepage

https://git-scm.com/

Opening GIT

Important

GIT Bash

check version

Tip

$ git - -version

Configuration

- Configure your *username* and *email*

Note

$ git config - -global user.name “Ana Devops”

Note

$ git config - -global user.email “anadevops@gmail.com”

Help

Important

$ git help config

Important

$ git config –help

Example

Developer Ana :

- She is working in her new project in a file call clients.txt

Creating a new repository

Note

```
$ git init project
```

Note

```
$ cd myproject
```

Tracking-Staged clients.txt

start tracking *holding zone* . Zone already to be commit

Note

$ git add clients.txt

Note

$ git status

In color *green*

unTracking - unstage clients.txt

Note

$ git restore - -staged clients.txt

Note

$ git status

In color *red*

Commit command

Remember: put in stage area with add

Note

$ git add clients.txt

Note

$ git commit -m “first commit”

Commit is the keyword that permanently logs changes

Exercise

Make 2 or 3 commits - Use $ git diff to see differences

Return to the previous commit

Note

$ git log - - oneline

Note

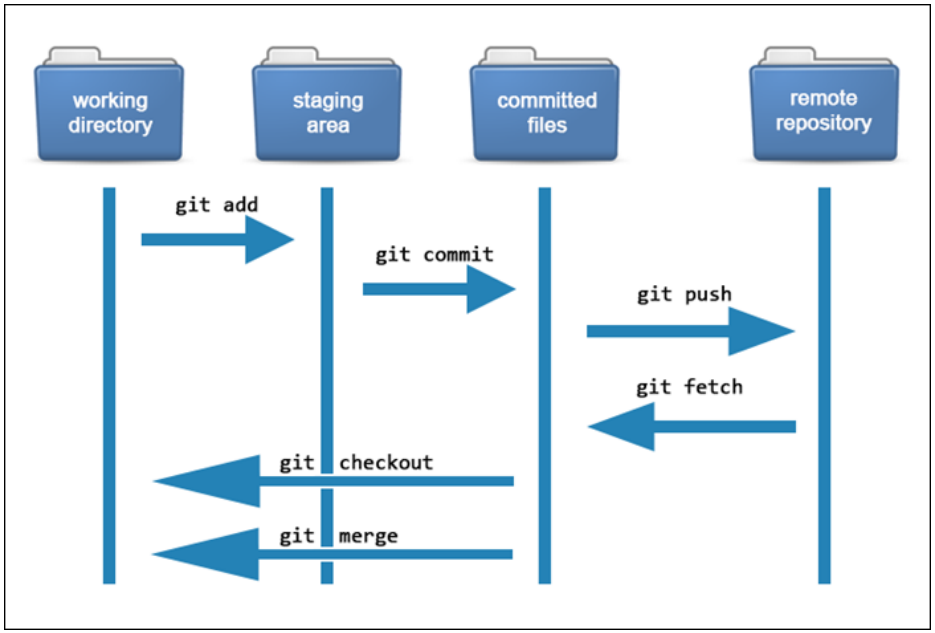
$ git revert ID

write :qa!

Note

$ git log - - oneline

GIT Workflow



Branches

Note

$ git branch potential

into branch potential: checkout

Note

$ git checkout potential

Note

$ git branch -l

branch vs master

make changes

Note

- $ git add .
- $git commit -m “potential clients”
- $ git status $ git checkout master

Merge branches

- put the changes from **potential** into **master**

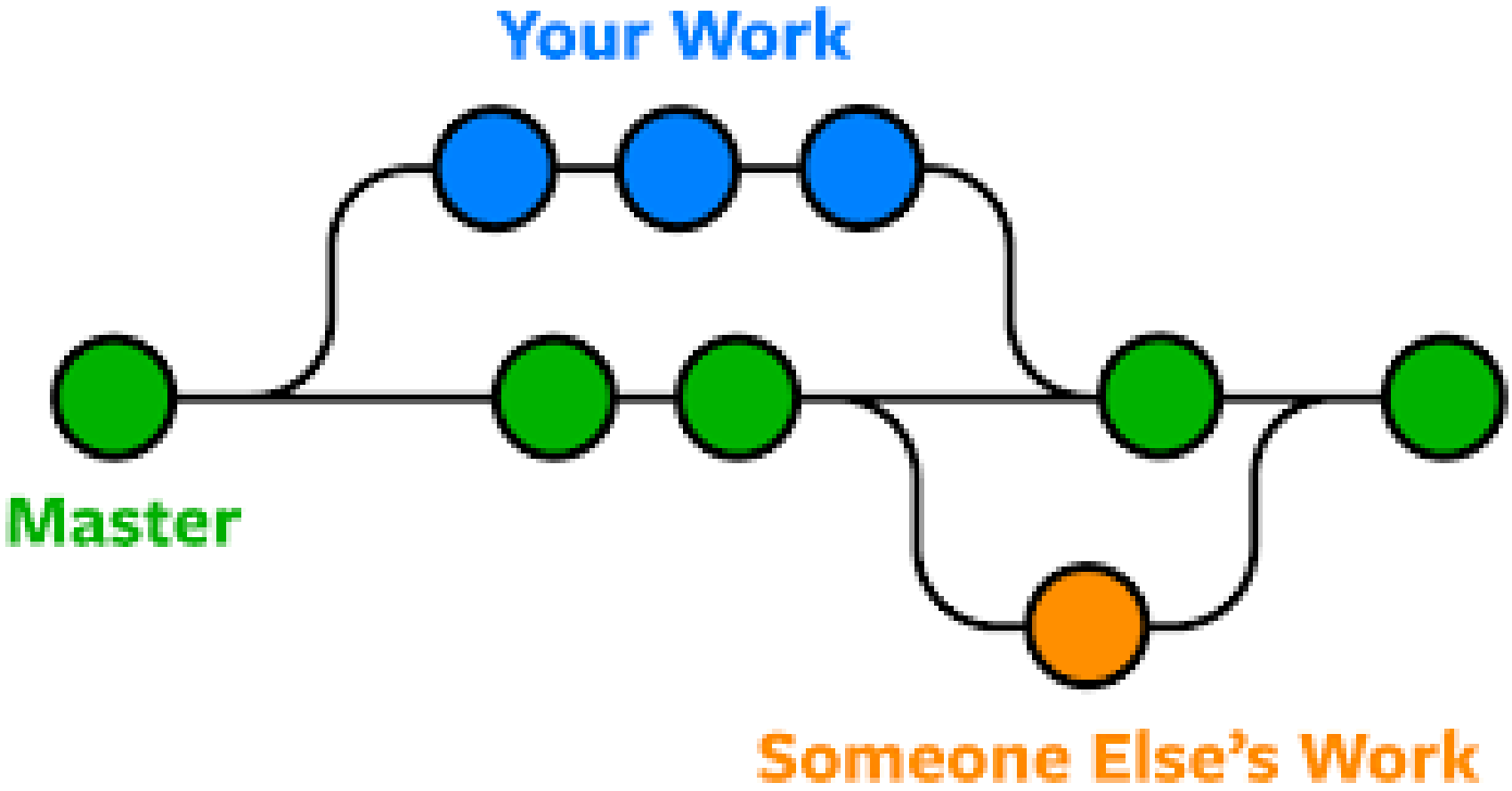
Note

$ git checkout master

Note

$ git merge potential

Merge details



Deleting potential branch

Note

$ git branch -d potential

Note

$ git branch -l

REMOTE repository

GitHub - GitLab

Commit all your changes and **PUSH** your **branch** to REMOTE

Note

$ git pull origin master

- Change the name master to main

Note

- $ git branch -M main
- $ git remote add origin URL
- $ git push -u origin main

Tracking an existing remote project

Note

$ git clone URL

Command that clone all the files from the repository and include in my working directory

- $ ls -la
  - Viewing all the information about the remote repository
- $ git remote -v
  - list the information of the repository
- $ git branch -a
  - list all branches local and remotely

Tracking: Cloned project

- make changes in the code:
  - add, commit, diff, status
- **push**

> **!** Important
>
> *REMEMBER*: Multiple developers

PULL any changes that have been made since the last time that we cloned the repository

- $ git pull origin master
- $ git push origin master

DAILY RUTING

- $ git branch DOTHIS
- $ git checkout DOTHIS
- make modifications
- $ git add . - $ git commit -m “changes”
- $ git checkout master
- $ git pull origin master
- $ git merge DOTHIS
- $ git push origin master
- $ git branch -d DOTHIS

Exercises

Explore commands

- $ git fetch
- What is difference between git fetch and git pull?

Aditional Documentation

https://git-scm.com/

https://education.github.com/git-cheat-sheet-education.pdf.
https://about.gitlab.com/images/press/git-cheat-sheet.pdf

https://ndpsoftware.com/git-cheatsheet.html

- Book Reference

Chacon, Scott, and Ben Straub. 2014. *Pro Git*. Apress.