GIT

Pamela Solano

Goals

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

References

Pro Git book, written by Scott Chacon and Ben Straub. 2014.

(Chacon and Straub 2014)

https://git-scm.com/book/en/v2

Target

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

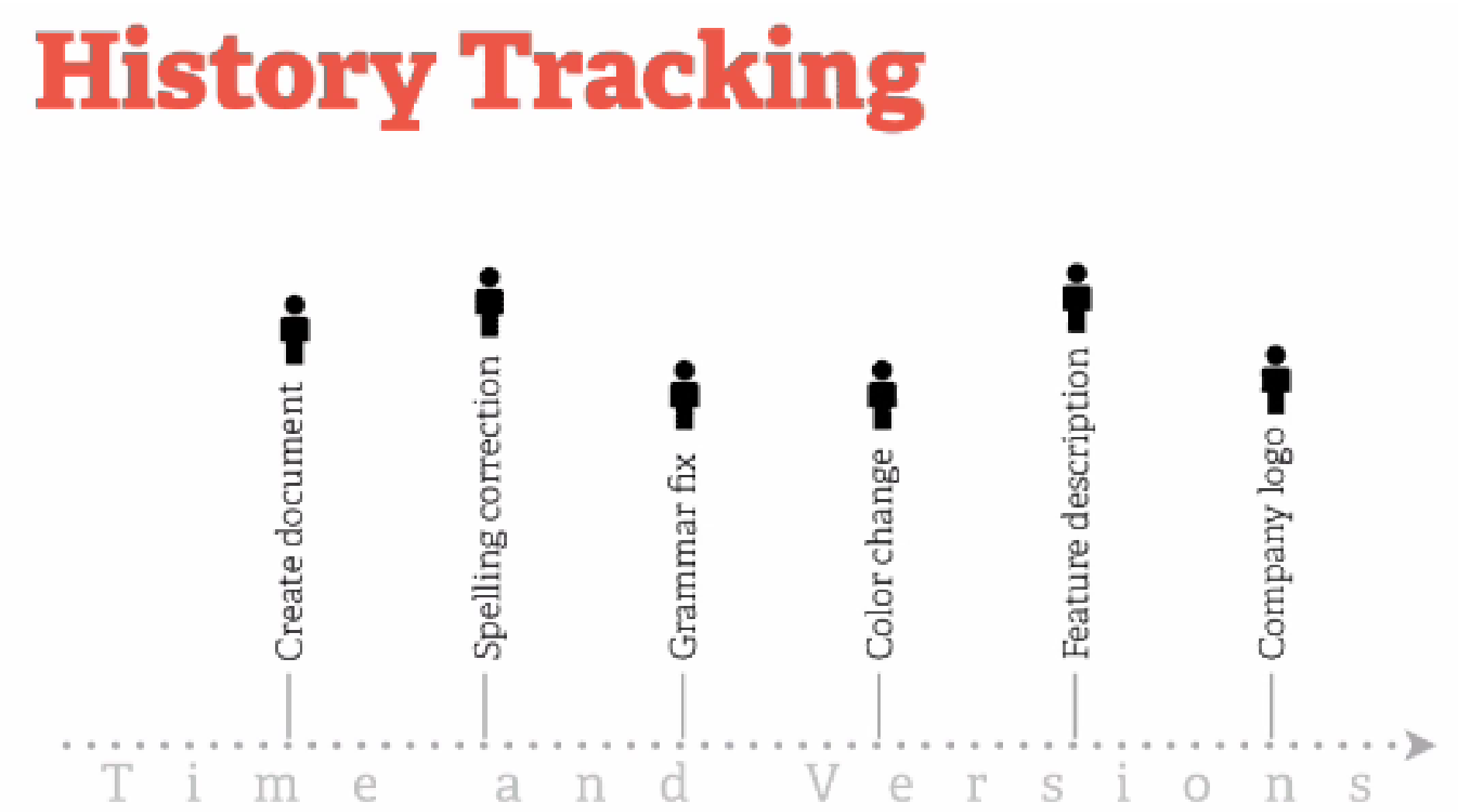
Your job and daily task

- Software developer
- Designer
- Someone who writes code

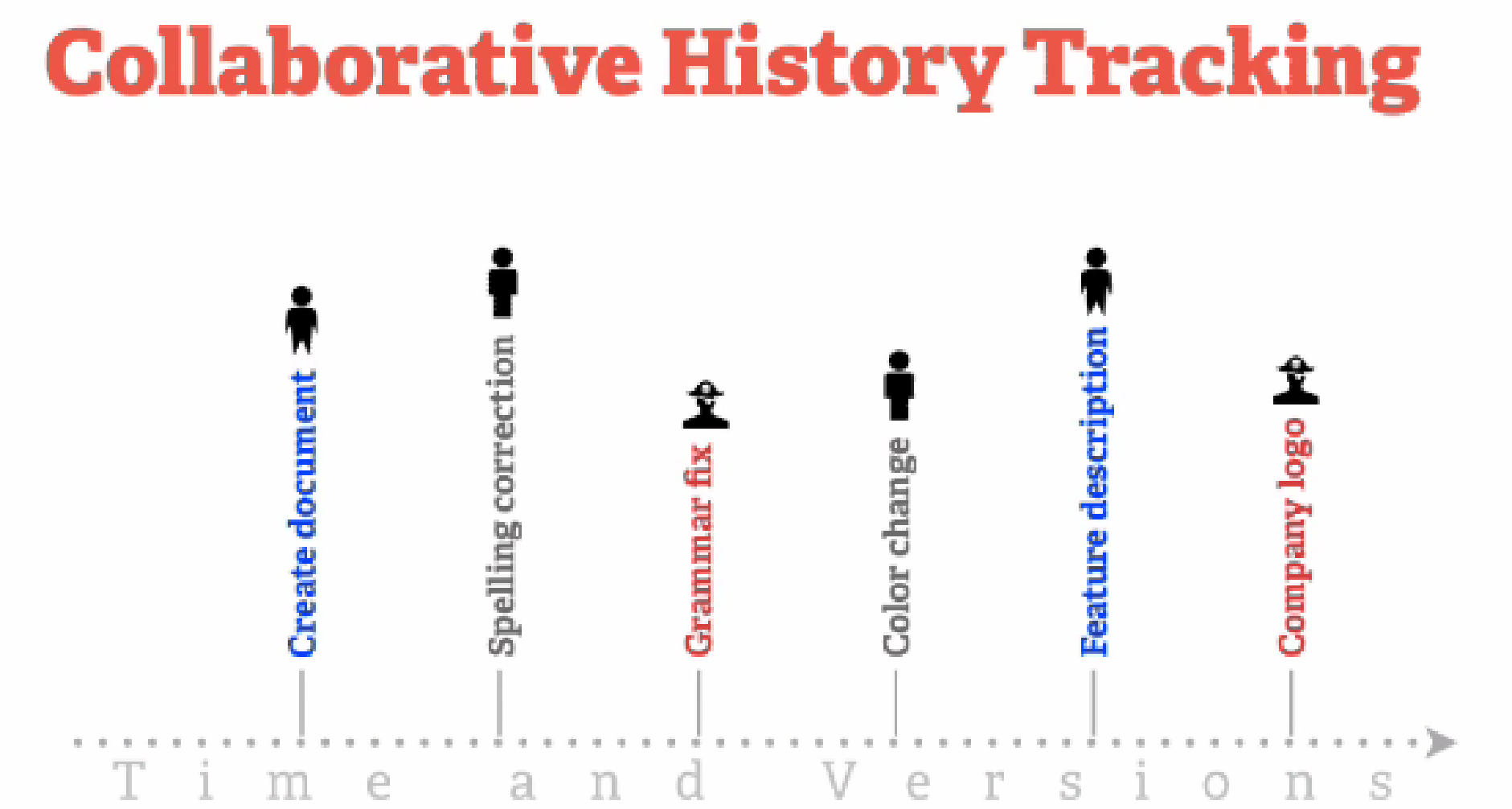
your task

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

Personal changes

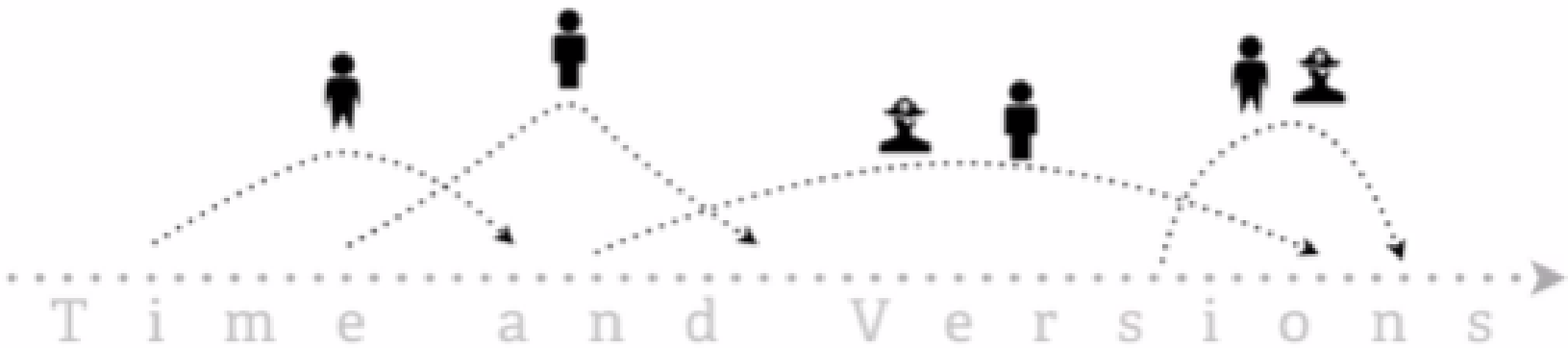


Team work


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

! Important

When-Why-What-Who are the goals *VERSION CONTROL*

Why GIT?

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

Why GIT? continuation

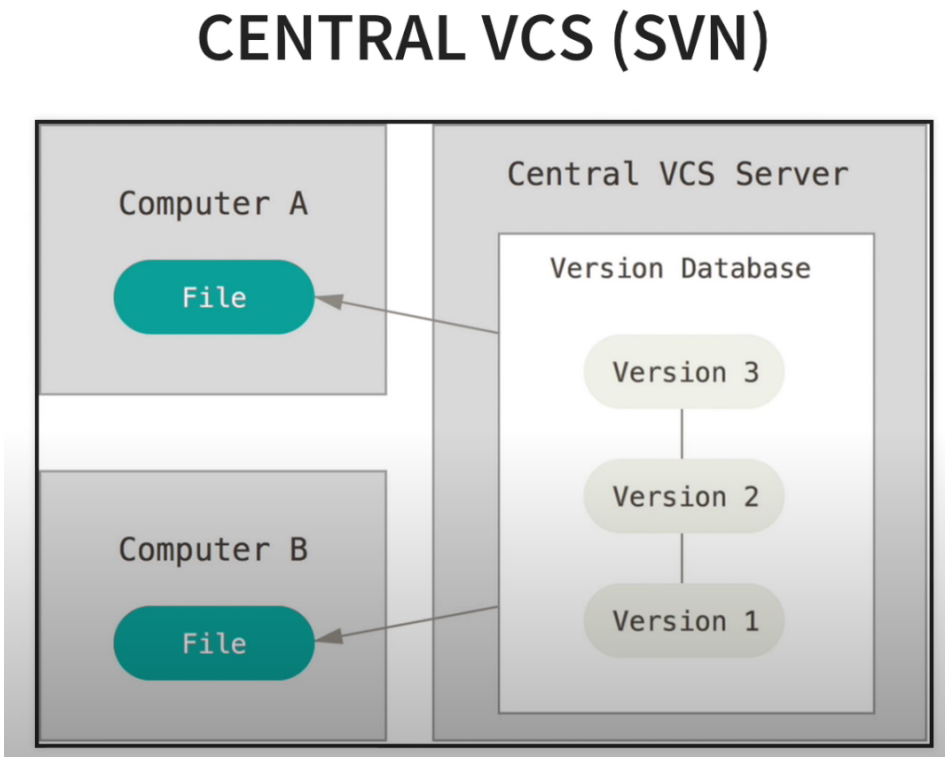
- GIT facilitates **collaborative changes** to files.
- Not just one person bringing modification and for all the team at the same time.
- People simultaneous change files, working at the same time about an idea.
- GIT is easy to use for
  - any type of knowledge worker.

Types of version control

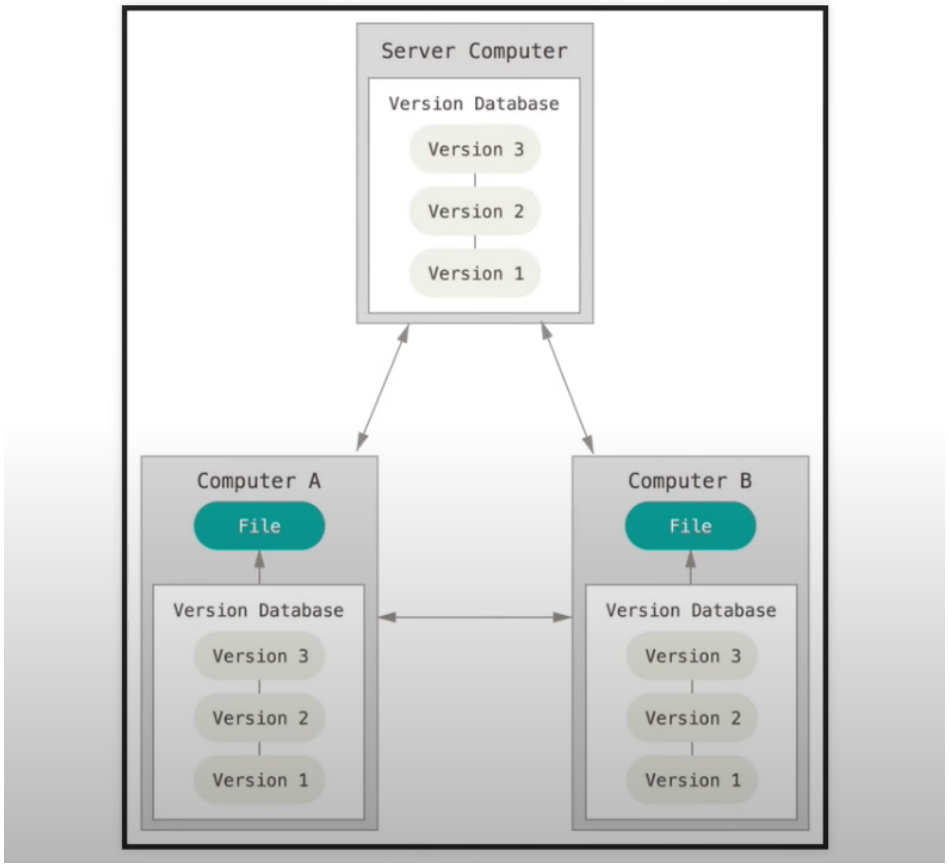
What is the difference between *CENTRAL* and

DISTRIBUTED Version Control System.

Without GIT: SVN



DISTRIBUTED VCS (GIT)



GIT

Version control

- Local GIT
  - Simple commands.
- *Distributed*
  - Connectivity not required
  - Simple software
- *Easy* $\rightarrow$ commands can be learnt *progressively*.

Summary

Important

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

Tip

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

Get going with git

- How to install
- Setup
- Configure
- 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

- Google is your friend

Example

Developer Ana :

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

Creating a new repository

Note

$ git init project

- If the directory exists use only

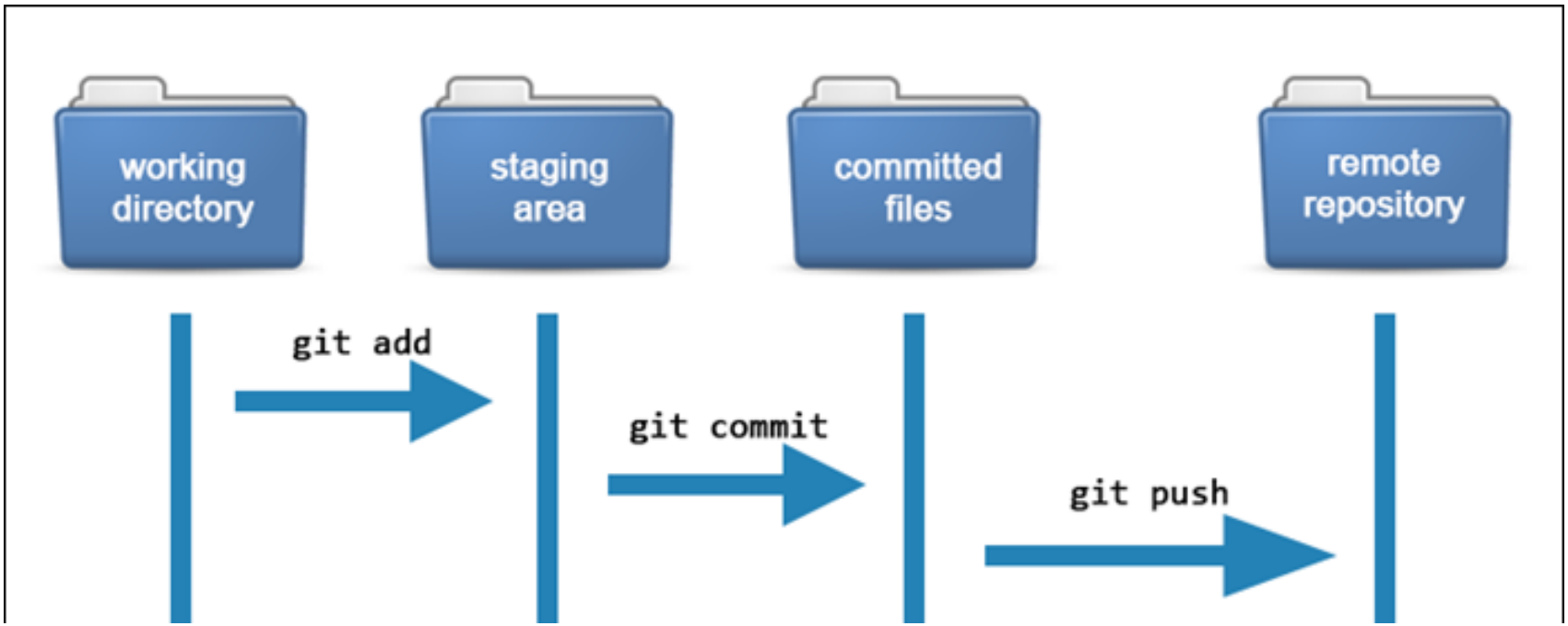
Note

$ git init

Note

$ cd project

Git workflow



Tracking-Staged clients.txt

Start tracking *holding zone*. Zone ready to be committed

Note

$ git add clients.txt

Note

$ git status

In color *green*

In your Git Bash

unTracking - unstage clients.txt

Note

$ git restore - -staged clients.txt

Note

$ git status

In color *red*

In your Git Bash

Commit command

Remember: put in stage area with add before commit

Note

$ git add clients.txt

Note

$ git commit -m “first commit”

Commit is the keyword that permanently logs changes

Exercise

- Make 3 commits
- Use $ git diff to see differences when relevant

Return to the previous commit

Note

$ git log - - oneline

Note

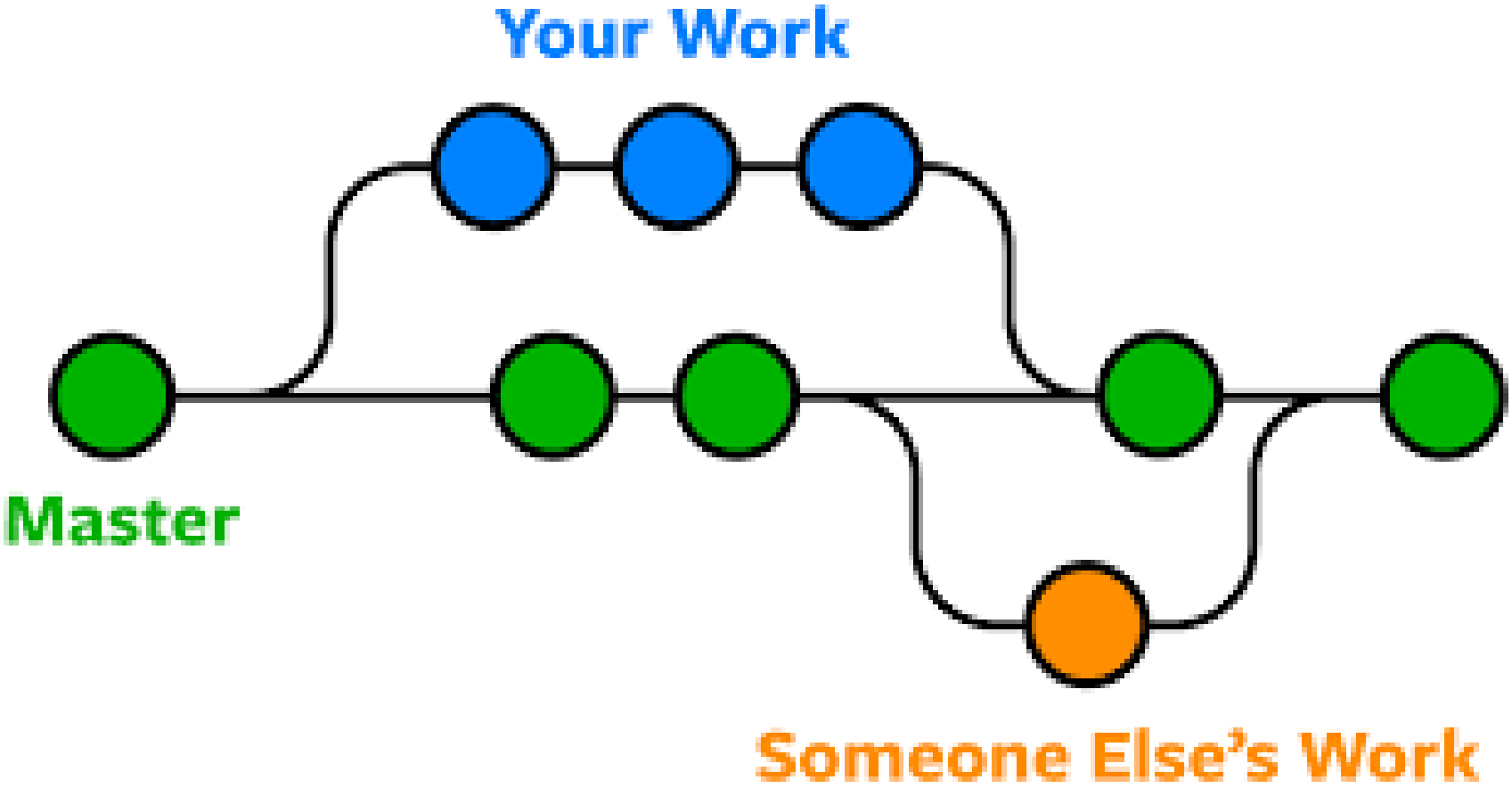
$ git revert ID

write :qa! (VIM's command)

Note

$ git log - - oneline

Our objective



Branches

A **Branch** is a version of your repository. An independent line of development.

> **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”

Master branch still old version

- $ git checkout master

Merge branches

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

Note

$ git checkout master

Note

$ git merge potential

Deleting potential branch

Note

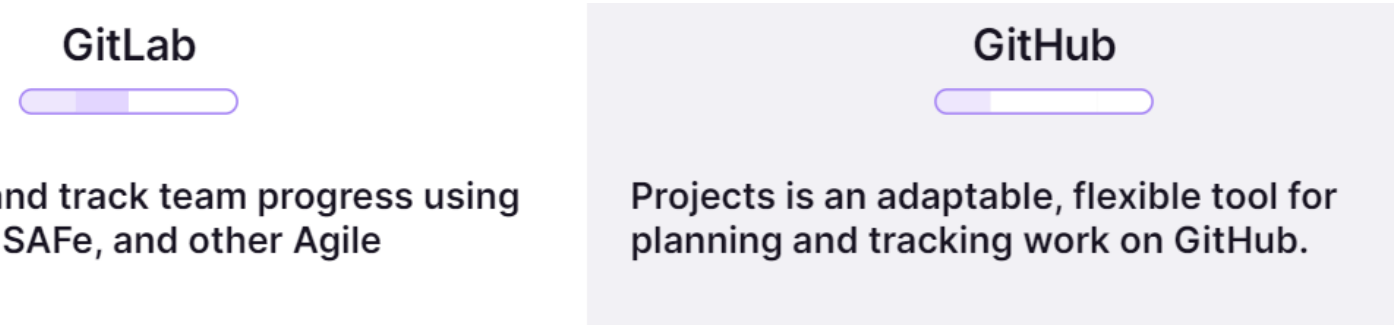
$ git branch -d potential

Note

$ git branch -l

Webservices

REMOTE repositories



REMOTE repository

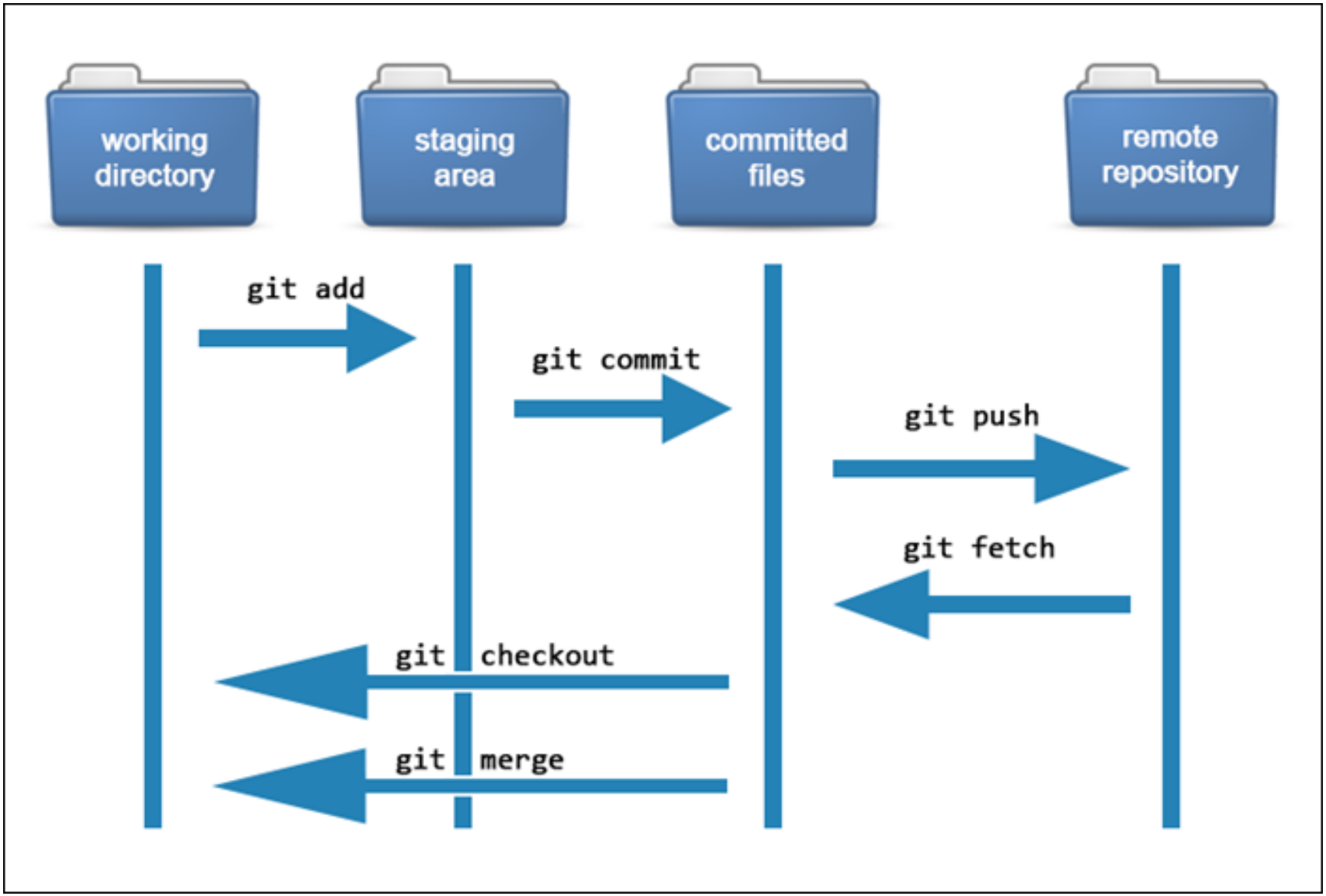
GitHub - *GitLab* your free WEBSERVICES

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

Note

- $ git remote add origin URL
- $ git push -u origin master

GIT workflow details



Tracking an existing remote project

Note

$ git clone URL

Command that clones all the files from the repository and includes in my working directory

- The **url** from GitLab or GitHub

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

Common commands

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

Wins-Benefits with GIT

- Focus on content.
- OPT in, not OPT-OUT: GIT's idea of the staging area.
- OPEN, not LOCKED, let people make contibution, **open source** project, free.
- DISTRIBUTED, not centralized.
- Exchange of code. WEBSERVICES like **GitHub** or **GitLab**.
- Focus on the PEOPLE, NOT TOOLS
- JOURNAL, NOT BACKUP. Version control system - historical changes
- ANYWHERE, NOT JUST ONLINE: GIT works entirely **OFFLINE** mode

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.