

# ITM/SoftServe Ruby Crash Course

León Jaramillo

softserve

# What is SoftServe?

**[www.softserveinc.com](http://www.softserveinc.com)**

# Who am I?

# Introduction to the Crash Course

## Learning Goals

Learning and applying main concepts about Ruby programming.

Learning and applying basic concepts about Ruby on Rails programming.

Getting basic knowledge towards working in enterprise environments.

## Prerequisites

Basic knowledge on software programming (including algorithms and Object-Oriented Programming).

Basic knowledge about web development technologies (i.e., HTML, CSS and Javascript).

English knowledge in a basic-independent level (B1, at least).

## Once the course is finished, the student will be able to...

Code a small program using Ruby language.

Code and deploy a small web app backend using Ruby on Rails.

Engage in Ruby-related intermediate courses, Project Labs and internships at SoftServe.

# Course Work Plan

1. Introduction to Git and version control systems.
2. Introduction to Ruby and its programming environment.
3. Operators, conditional statements, and loops.
4. Methods, blocks, procs, and Lambdas.
5. Arrays and hashes.
6. Strings.
7. OOP basics.
8. Inheritance, modules and mixins.
9. Exception handling.
10. Basic web development.
11. Ruby on Rails introduction.
12. Web deployment.
13. Controllers and routing.
14. Action View and layouts.
15. Introduction to unit testing.



# Introduction to Git and version control *systems*



**What is SCM?**

**What is a VCS?**

**What is Git?**

# What will we talk about?

Source Control Management (SCM)

Version Control Systems (VCS)

VCS Types

Git

Best Practices

Branching Strategies

# What is Source Code Management (SCM)

## Source Code Management (SCM)

is a **set of practices** that programmers use to manage source code:

- Backup of assets
- Synchronization of work
- Undoing changes
- Tracking changes
- Code ownership
- Branching and merging

Why?

Coordinating developers

Editing shared pieces of code

Conflicts prevention and/or management

# What is Version Control Systems (VCS)

- **Version control** is tracking and managing changes to software code
- **Version control systems (VCS)** are software tools that help software teams to perform version control
- VCS aim to **accelerate** the software development process
- VCS keeps **track** of every modification done by the development team on their codebase

## Which are its benefits

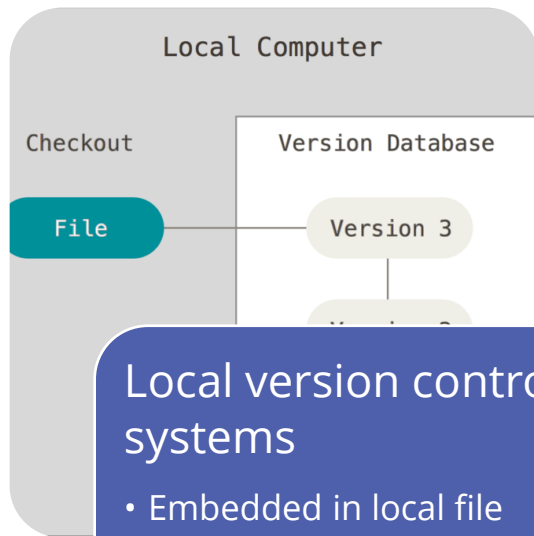
A complete long-term change history of every file.

Branching and merging

Traceability

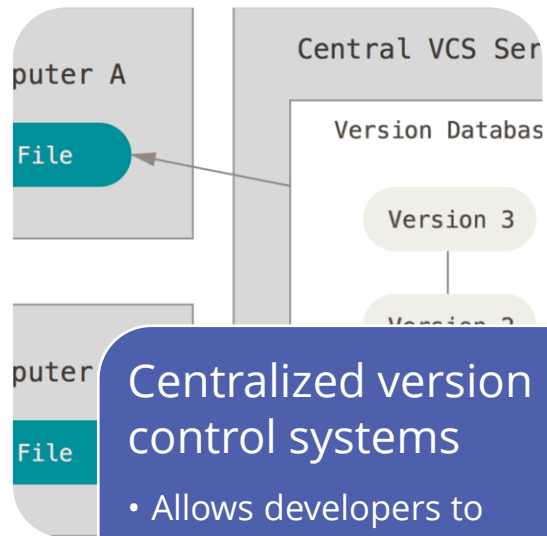
Offsite code backup

# Where did the VCS come from (shaping their types)?



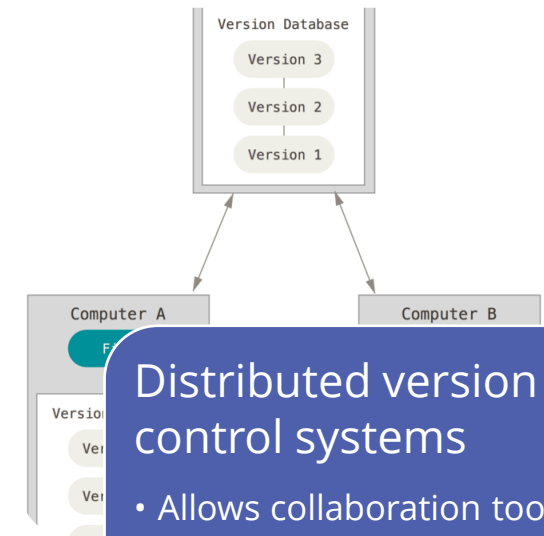
## Local version control systems

- Embedded in local file system.
- Often patch based (i.e, RCS).
- Prone to data loss.



## Centralized version control systems

- Allows developers to collaborate between them (i.e, CVS, SVN, Perforce).
- One single server hosts the whole repo.
- Makes a single point of failure + network overhead.



## Distributed version control systems

- Allows collaboration too (i.e, Git, Mercurial).
- Each client has a copy of the repo in their machine.
- Allows several ways to collaborate, in addition to different workflows.

Images from git-scm.com

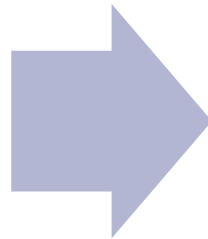
**softserve**

# What VCS can we find out there?

Tool	Architecture	Conflict resolution	Development status	URL
Git	Distributed	Merge	Active	<a href="https://git-scm.com/">https://git-scm.com/</a>
Mercurial	Distributed	Merge	Active	<a href="https://www.mercurial-scm.org/">https://www.mercurial-scm.org/</a>
SVN	Centralized	Merge or lock	Active	<a href="https://subversion.apache.org/">https://subversion.apache.org/</a>
CVS	Centralized	Merge	Maintenance only	<a href="https://www.nongnu.org/cvs/">https://www.nongnu.org/cvs/</a>

And more...: [https://en.wikipedia.org/wiki/List\\_of\\_version-control\\_software](https://en.wikipedia.org/wiki/List_of_version-control_software)

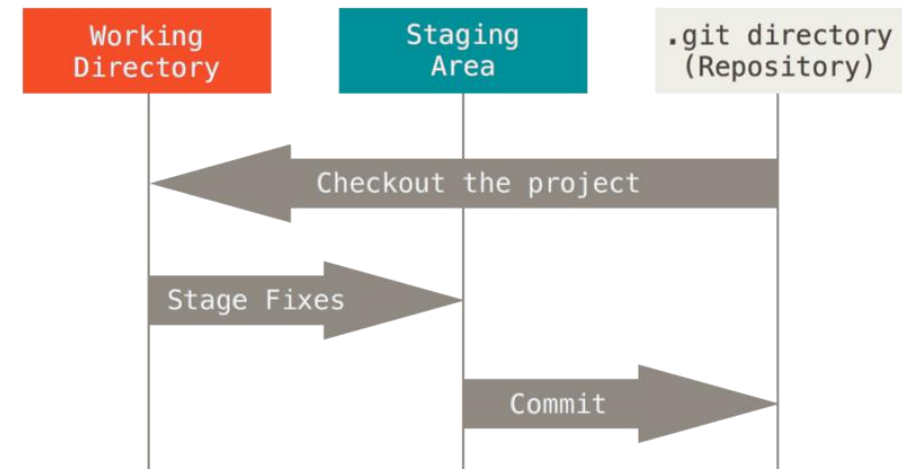
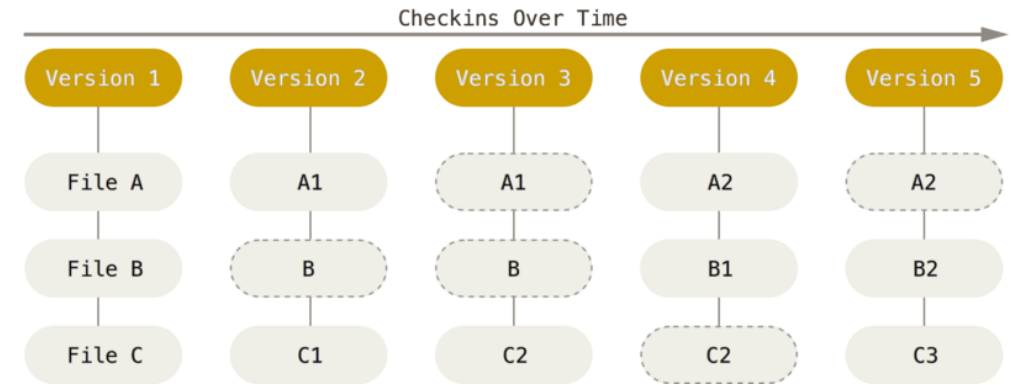
So, where are GitHub,  
GitLab or Bitbucket?



They're not VCS, but VCS  
hosting services.

# What is Git?

- **Git** is a Distributed Version Control System (DVCS).
- It is based on **snapshots**, so every commit saves the state of the project at a given moment.
- Most of the operation done with Git are **local**.
- Everything in Git is **checksummed** using SHA-1.
- Locally, at Git, every files are in one of three states: **modified**, **staged** or **committed**.



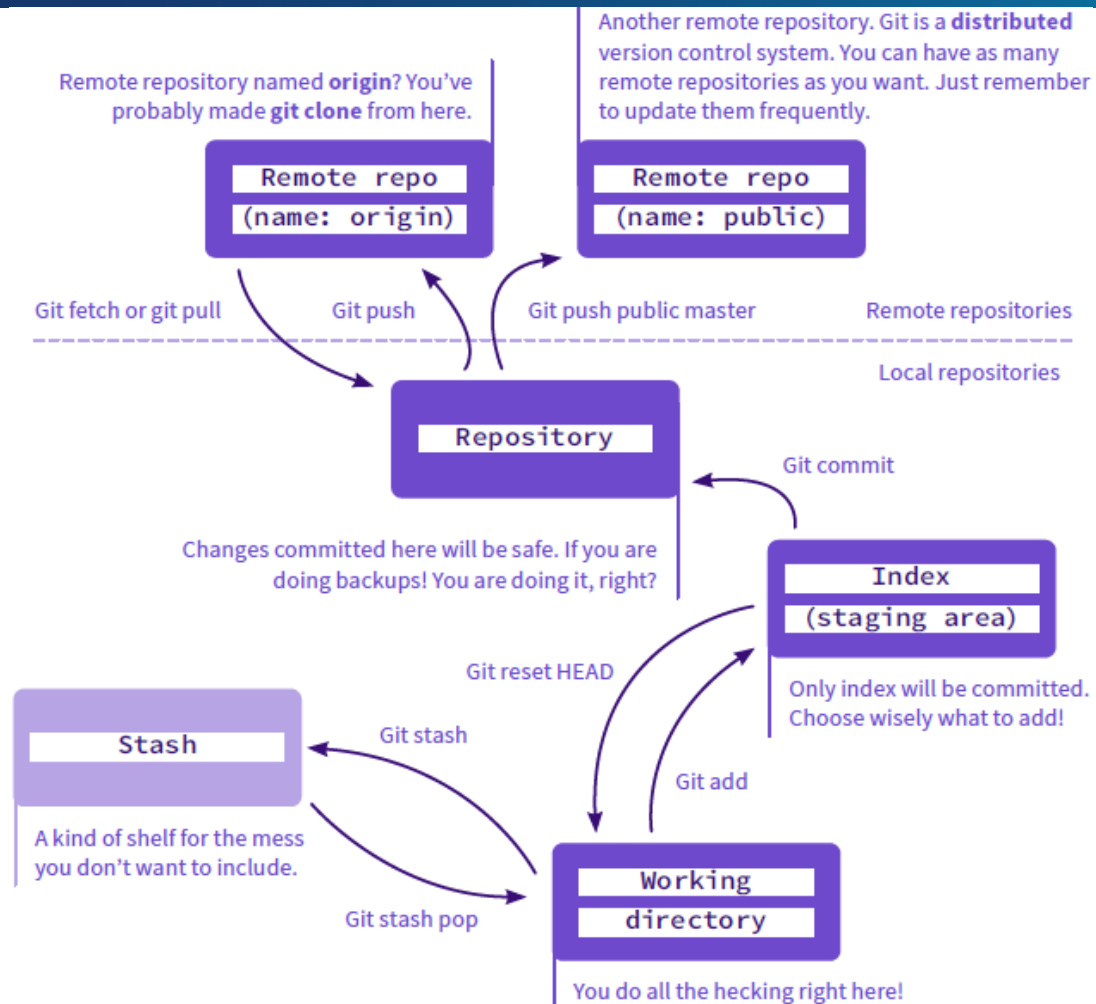


# So, what is GitHub?

- **GitHub** (<https://github.com/>) is a hosting service for software development and version control using Git.
- Comprises different **services** besides DVCS, notably, GitHub Actions or GitHub Pages.
- Nowadays is owned by **Microsoft**.
- Hosts **millions** of public and private **repositories**.
- It is just one of several **alternatives**.



# The zoo of working areas



From: GitLab Cheat Sheet

# Let's try a little bit...

```
git clone [repository-URL]
```

Clones remote repository to the current local directory

```
git status
```

Shows modified and staged files

```
git add [.|file-name]
```

Adds a file or files to the staging area

```
git commit -m "[message]"
```

Commits the staged files

```
git push [alias] [branch]
```

Pushes local commits to a remote branch

# Let's try a little bit...

```
git init
```

initializes the current directory as a Git repository

```
git diff
```

Diff of what is changed but not staged

```
git reset [file-name]
```

Unstages the specified file, given that it's already at the staging area

```
git branch
```

Lists all the branches in your repository

```
git branch [name]
```

Creates a branch with the specified name

# Let's try a little bit...

```
git init [folder name]
```

initializes a Git repository in a new folder

```
git diff --staged
```

Diff of what is staged but not committed

```
git checkout [branch]
```

Switches to another branch and

```
git log
```

Show all the commits in the local branch history

```
git branch -d [branch]
```

Removes the selected branch

# Version Control best practices

Commit often

Commit related changes

Do not commit half-done work

Write good commit messages (making detailed notes if needed)

Ensure you're working from latest version

Review changes (and test) before committing

Use branches

Agree on a workflow (and branching strategy)

From: Atlassian and Git Tower

**softserve**

# Git Branching Strategies

A branching strategy sets rules about how a team of developers write, merge and deploy code using a VCS



GitFlow

GitHub Flow

Trunk-Based Development

Scaled Trunk-Based Development

*Custom workflows*

# Useful Links

- Git and GitHub for Beginners - Crash Course: <https://www.youtube.com/watch?v=RG0j5yH7evk>
- Git Branches Tutorial: <https://www.youtube.com/watch?v=e2IbNHi4uCI>
- Git for Professionals Tutorial: [https://www.youtube.com/watch?v=Uzsj\\_k0DGsg](https://www.youtube.com/watch?v=Uzsj_k0DGsg)
- Advanced Git Tutorial: <https://www.youtube.com/watch?v=qsTthZi23VE>
- Git Cheat Sheet (GitLab): <https://about.gitlab.com/images/press/git-cheat-sheet.pdf>



# Homework

1. Create a GitHub account (if you don't have it yet).
2. Install Git (<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>).
3. Create a remote repository using GitHub, it must include a LICENSE and a README files.
4. Get it to your preferred workspace.
  - If you don't have a preferred workspace, try using Visual Studio Code and its console.
5. Create a basic HTML file (a "Hello world" might work) and push it to the remote.



# COLOMBIA

# DC

Thanks! Any question?

softserve