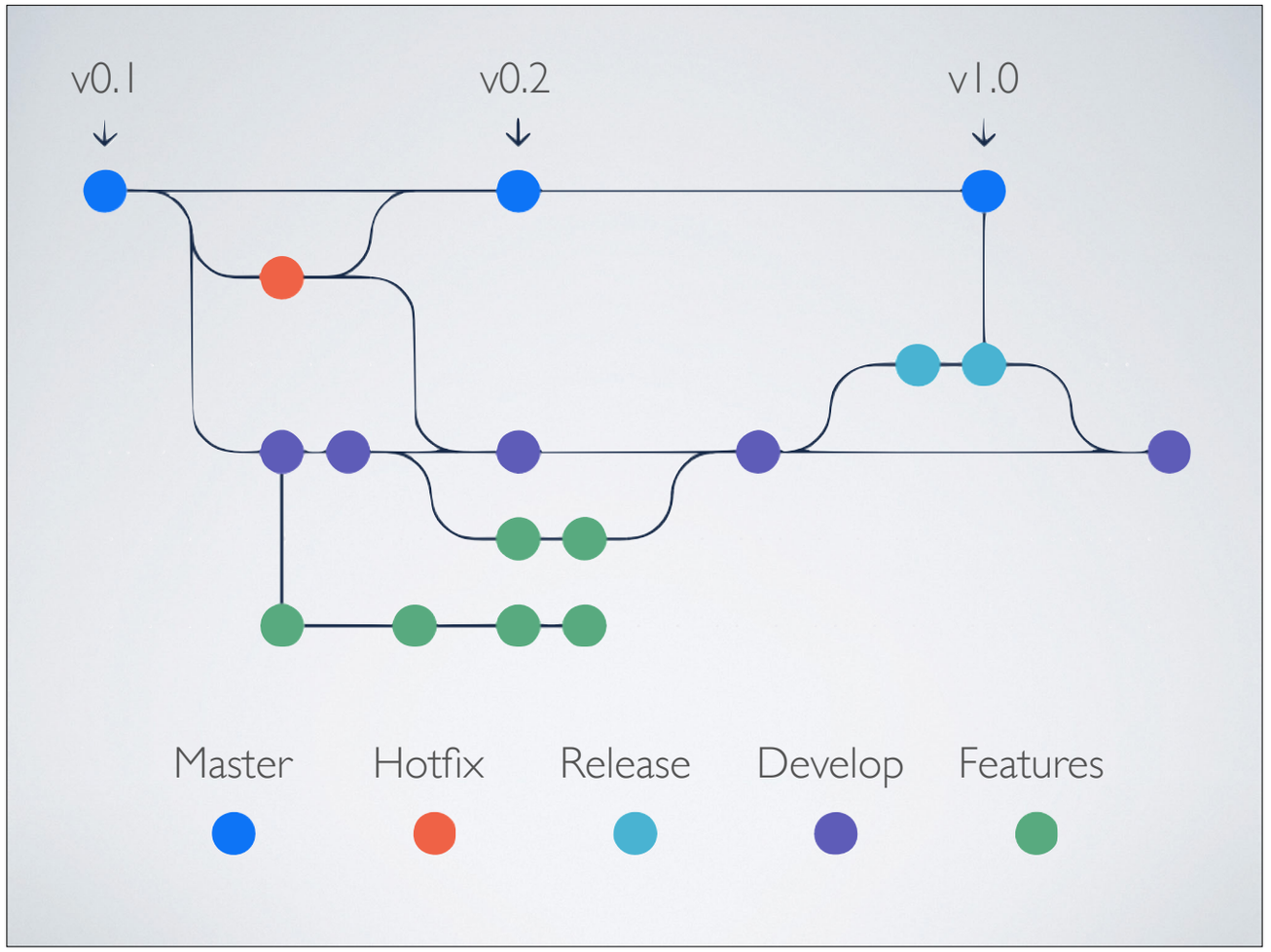
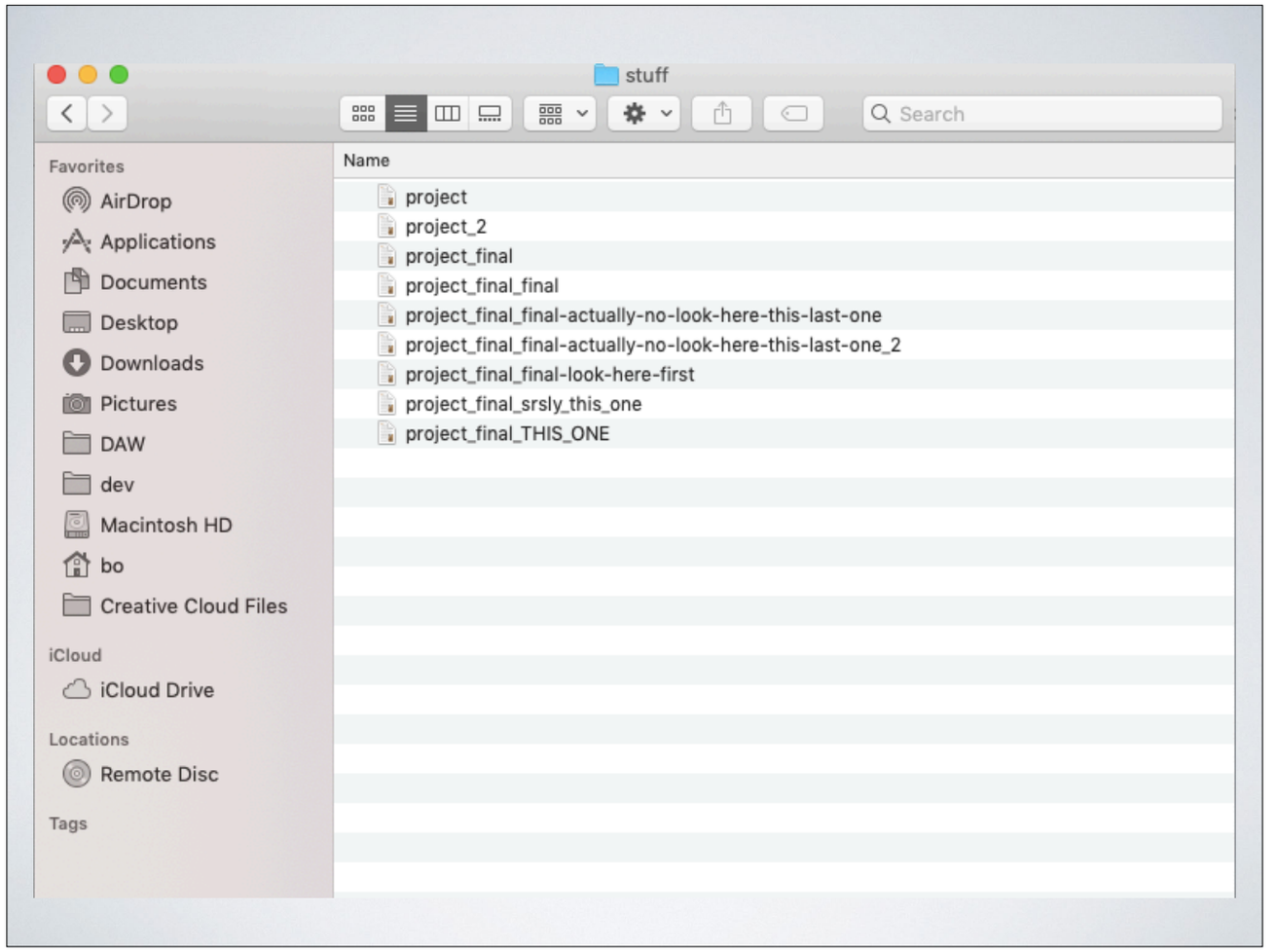
VERSION CONTROL


v0.1
v0.2
v1.0
Master
Hotfix
Release
Develop
Features

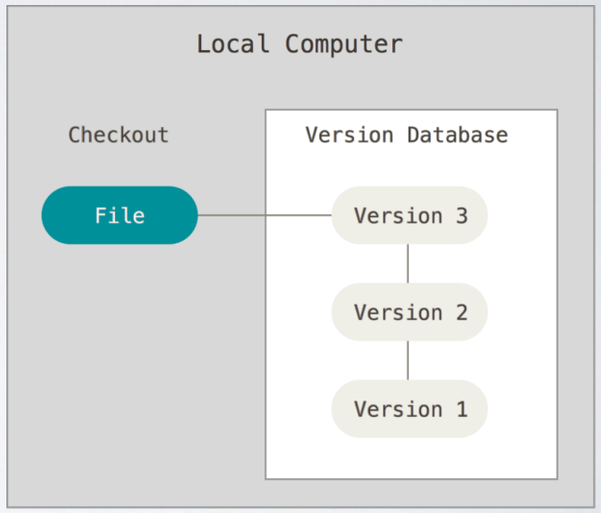
VERSION CONTROL

- Es un sistema que registra los cambios de un archivo o carpeta de archivos en versiones con el fin de poder consultarlas en el futuro

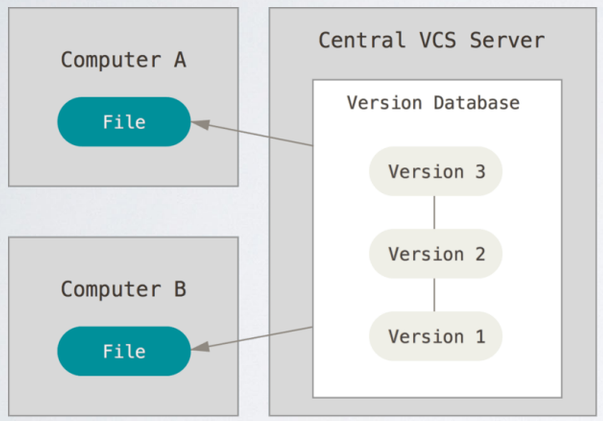

stuff
Search
Favorites
AirDrop
Applications
Documents
Desktop
Downloads
Pictures
DAW
dev
Macintosh HD
bo
Creative Cloud Files
iCloud
iCloud Drive
Locations
Remote Disc
Tags
Name
project
project_2
project_final
project_final_final
project_final_final-actually-no-look-here-this-last-one
project_final_final-actually-no-look-here-this-last-one_2
project_final_final-look-here-first
project_final_srsly_this_one
project_final_THIS_ONE

LOCAL VERSION CONTROL SYSTEM

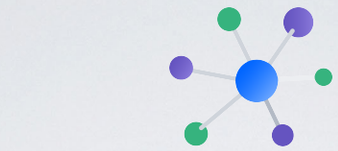
- BD simple que guarda todos los cambios de un archivo
- EJ: RCS, guarda "parches" de un documento en un formato diferente. Re-crea una versión poniendo "parches" sobre el documento



CENTRALIZED VERSION CONTROL SYSTEM



- BD en un servidor único con todos los archivos versionados
- Varios clientes pueden consultar esas versiones
- Ej: CVS, Subversion, Perforce



Ventajas
(contra un LVCS)

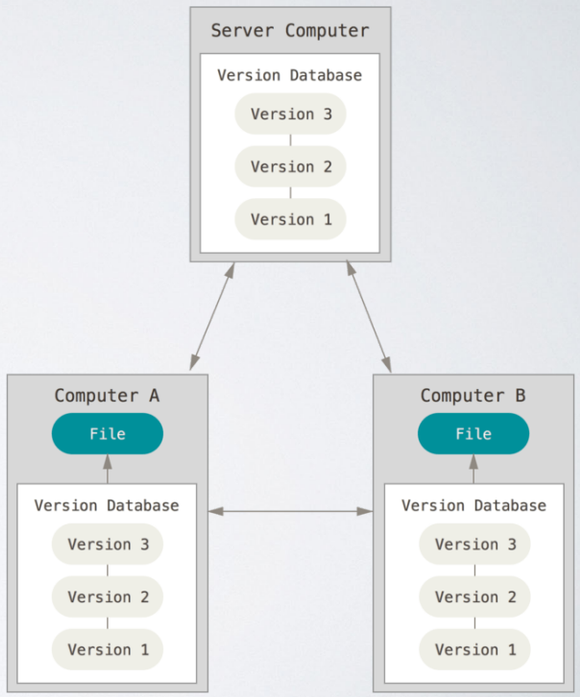
- Todos conocen, hasta cierto punto, que hacen los demás: contribuciones particulares
- Administración de permisos centralizada, sencilla

Desventajas

- Hay un *solo punto de falla*
- Si el servidor se daña, no existen respaldos adecuados, se pierde la historia completa a excepción de las versiones locales

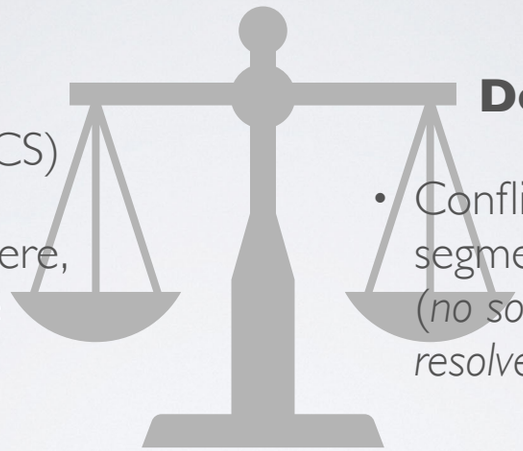
DISTRIBUTED VERSION CONTROL SYSTEM

- Cada cliente tiene un clon exacto del repositorio central incluyendo el historial completo
- Se puede colaborar de manera simultánea y remota
- Ej: Git, Mercurial, Bazaar, Darcs



Ventajas
(contra un CVCS)

- Sí el servidor muere, cualquiera puede restaurarlo



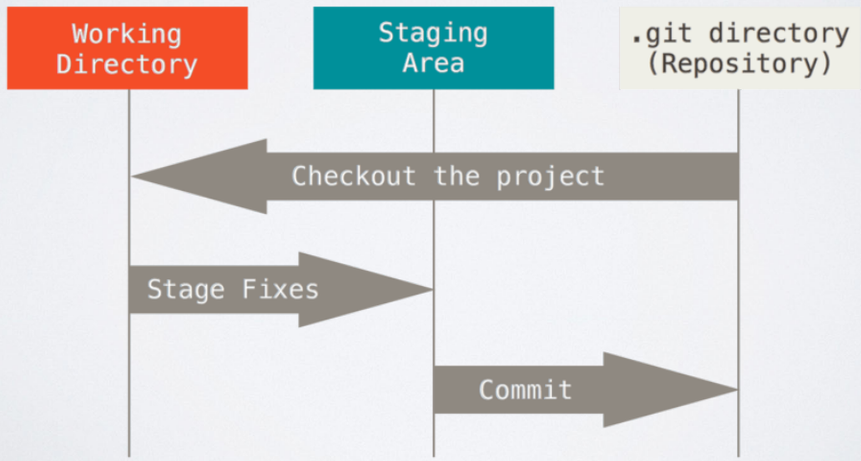
Desventajas

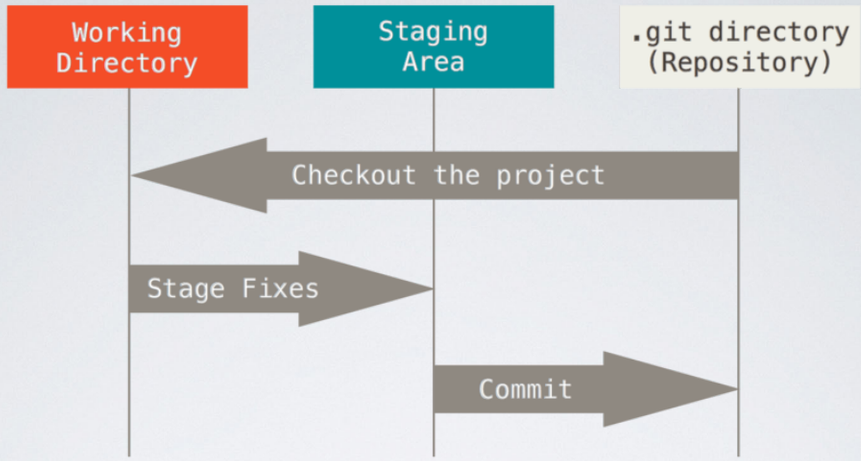
- Conflictos entre mismos segmentos de código (*no son imposibles de resolver*)


git

- 1991-2002: BitKeeper para Linux Kernel
- 2005: Linus Torvalds crea Git
  - Rápido
  - Diseño simple
  - Permite desarrollo paralelo (ramificaciones ilimitadas)
  - Completamente distribuido
  - Efectivo (tiempo, tamaño) para proyectos grandes

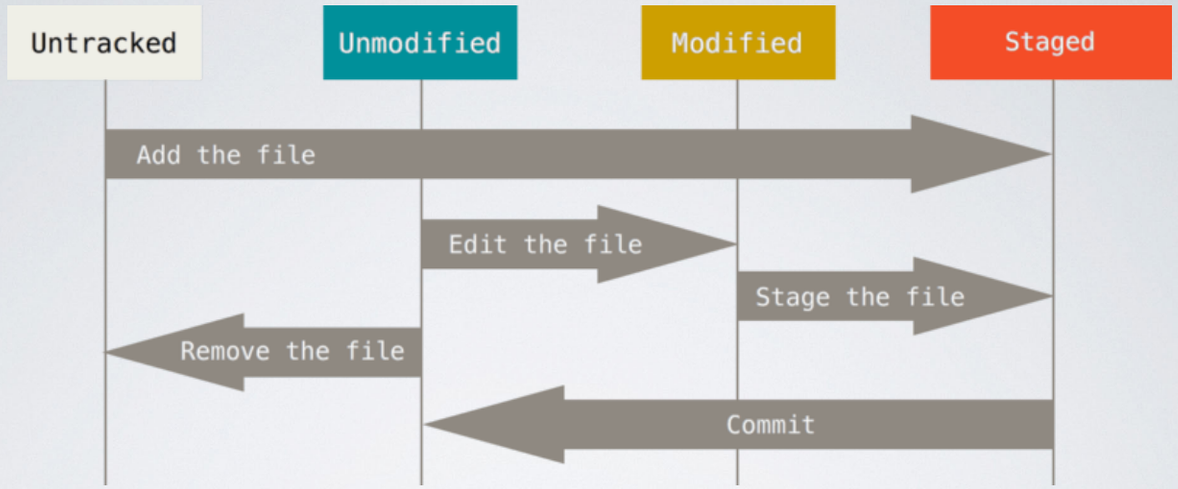
- Git tiene 3 estados
  - *Committed:* Los cambios están guardados localmente
  - *Modified:* Cuando los cambios están locales pero no los has guardado (committed)
  - *Staged:* Cuando haz marcado archivo(s) para ser agregados a la nueva versión (pre-commit)





- 3 secciones de un proyecto:
  - *Git directory*: Almacena metadata y la base de objetos del proyecto. Es la parte más importante de Git, y es lo que se copia cuando clonas el repo
  - *Working tree*: Es la copia de una versión del proyecto. Una vez que se obtiene del Git directory, se puede usar o modificar
  - *Staging area*: es un archivo, por lo general en tu Git directory, que almacena la información acerca de lo que irá en el siguiente commit

Proceso básico de GIT



1. Modificas archivos en el *working tree*
2. Decides que cambios se agregan al commit y se pasan a la *staging area*
3. Haces commit desde la staging area, se almacenan permanentemente en una versión en el *Git repository*

INSTALL AND SETUP

- https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
- https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup
- https://www.atlassian.com/git/tutorials/install-git

GIT CLONE <URL>

- Transfer protocols:
  - https

```
git clone https://github.com/project custom-name
```

  - ssh

```
git clone user@server:path/to/repo.git
```

SSH PUBLIC KEY

```
$ cd ~/.ssh
$ ls
authorized_keys2  id_dsa       known_hosts
config            id_dsa.pub

$ ssh-keygen -t rsa -b 4096 -C “your_email@example.com”
Generating public/private rsa key pair.
Enter a file in which to save the key (/Users/you/.ssh/id_rsa):
[Press enter]
Enter passphrase (empty for no passphrase): [Type a passphrase]
Enter same passphrase again: [Type passphrase again]

$ cat ~/.ssh/id_rsa.pub
COPIAR
```

https://help.github.com/articles/connecting-to-github-with-ssh/

https://www.ssh.com/ssh/keygen/

LABORATORIO 5

- Carrera de git:
  - Crea una cuenta en gitlab
  - git clone https://gitlab.com/yvone_enovy/hurry_up.git
  - cd hurry_up
  - Edita el archivo index.html y consulta index.css para referencia
    - Agrega un div que contenga a su vez, un div con tu nombre y uno con tus apellidos
    - Agrega la clase correspondiente al div exterior:
      - Sí la suma de los dígitos de tu matrícula es un número par: column, si es impar: row
      - Si el último digito es par: allinea al inicio en el mismo eje; si es impar, alinea al final en el mismo eje
      - Si tu primer nombre inicia con vocal: alinea al final en el eje contrario, si es consonante: alinea al inicio en el eje contrario

LABORATORIO 5

- git add index.html
- git commit -m "*nombre*"
- git pull origin master
- *Arregla conflictos*
- git push -u origin master

- Los **primeros cinco CORRECTOS**, tendrán 2 galletas y los **últimos tres** pierden el derecho a puntos extra en el parcial (Si ya tienen más de 2, no contarán)

LABORATORIO EXTRA

- git clone https://github.com/yvone/xml_and_xsd.git
- Agregar los siguientes nodos al xml y su validación correspondiente en el xsd
  - FichaDatos
    - Titulo
      - fechaExpedicion
      - creadoPor
    - Persona
      - nombre
      - apellidos
    - Contacto
      - correoElectronico
      - curp
    - **Direccion**
      - **calle**
      - **numero**
      - **colonia**
      - **CP**