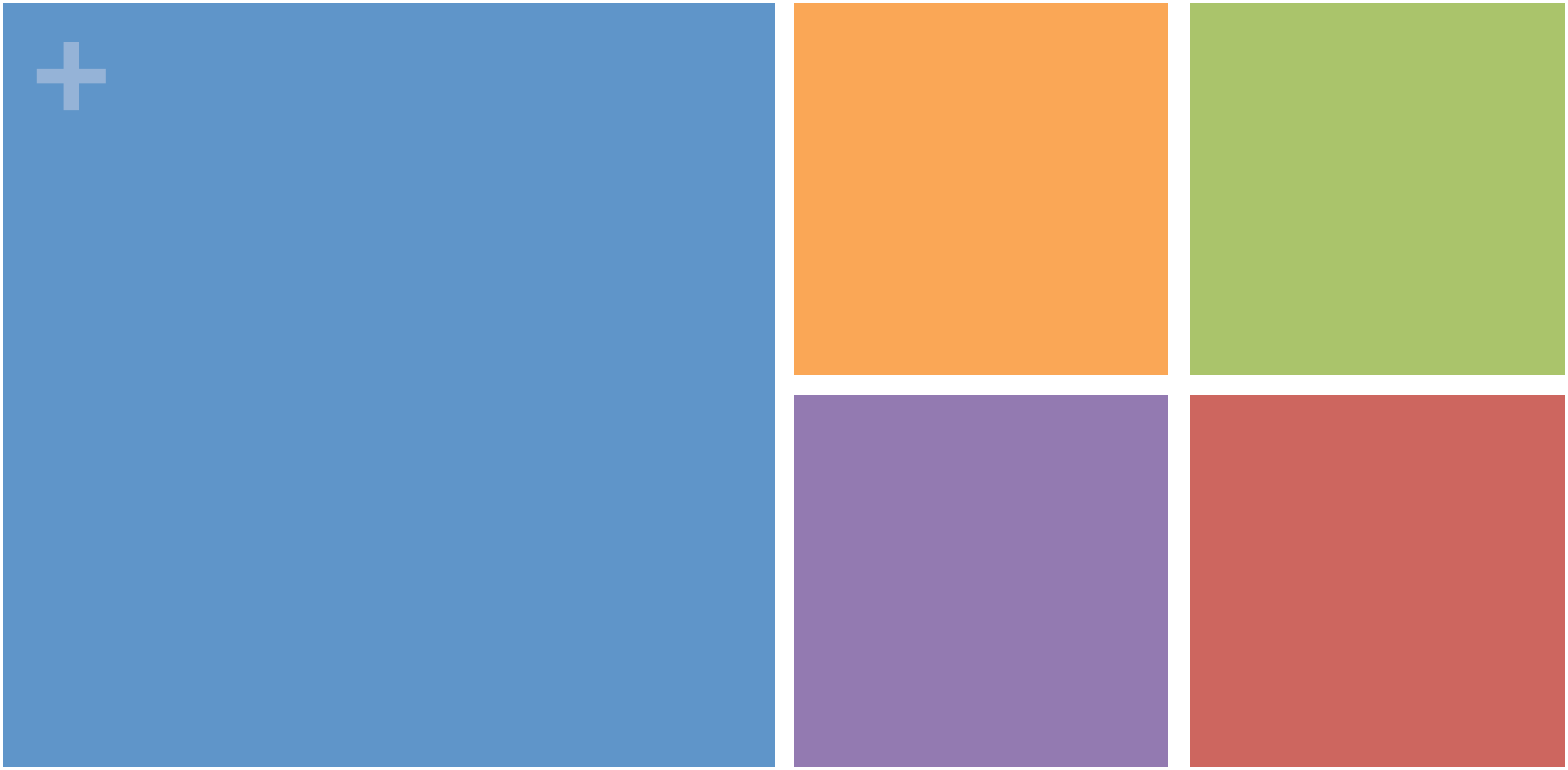


# Fundamentos de Programación

# *Tipos de Datos*

**Number:** incluye números con coma (Float) y enteros (Integer).

**String:** cualquier número de caracteres, por ejemplo "a", "uno", "uno 2 tres".

**Boolean**: puede ser verdadero o falso.

# Tipos de Datos

**Undefined:** cuando se intenta acceder a una variable que no existe, obtiene el valor indefinido. Lo mismo ocurrirá cuando se declara una variable, pero no se le ha dado un valor todavía. JavaScript lo inicializa por defecto como undefined.

**Null:** este es otro tipo de datos especial que sólo puede tener un valor, el valor nulo. Significa valor vacío. La diferencia con undefined es que si una variable tiene un valor nulo, esta definido pero como nulo.

# Control de versiones

*Sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo.*

- Mantiene un histórico del proyecto el cual nos permite volver atrás.
- Permite el desarrollo de forma concurrente.
- Sirve como Respaldo

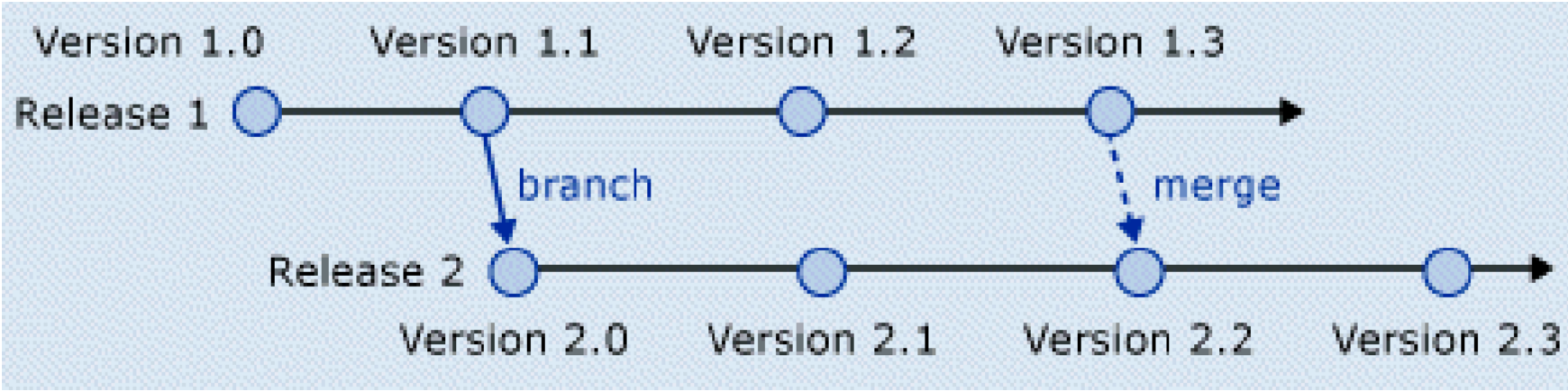
# Modelo versionado

- Se almacenan  los archivos en un repositorio.
- Checkout permite obtener el archivo para leer o escribir
- Checkin devuelve al repositorio los archivos modificados creando una nueva versión

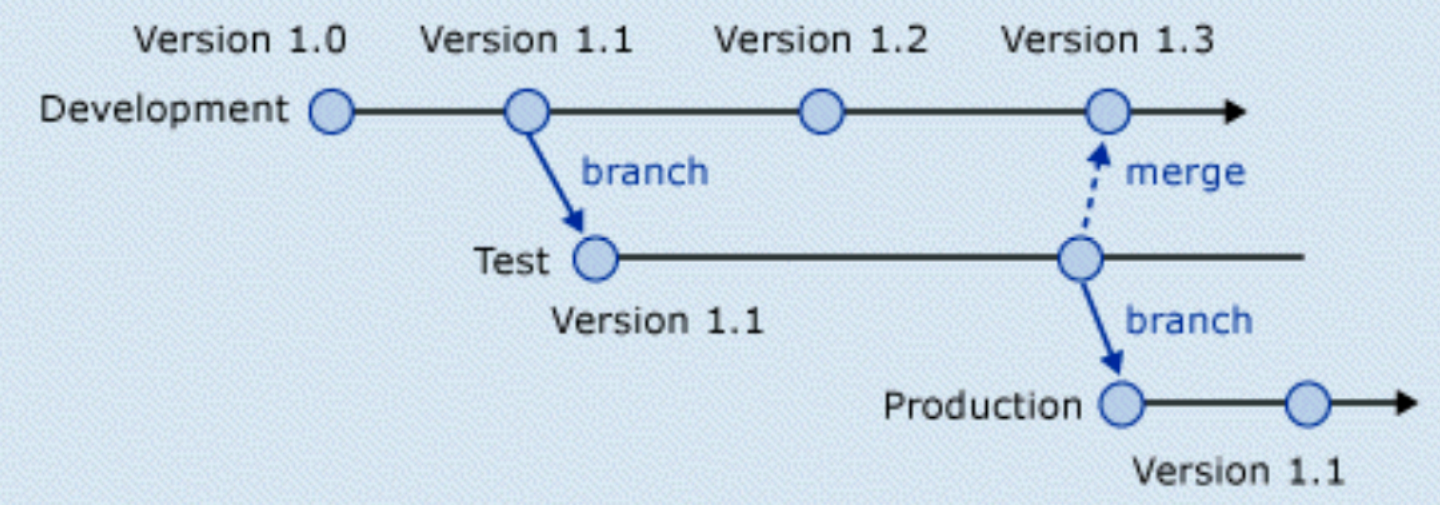
Tres formas de evolución:

- Versionado
- Merge
- Branch

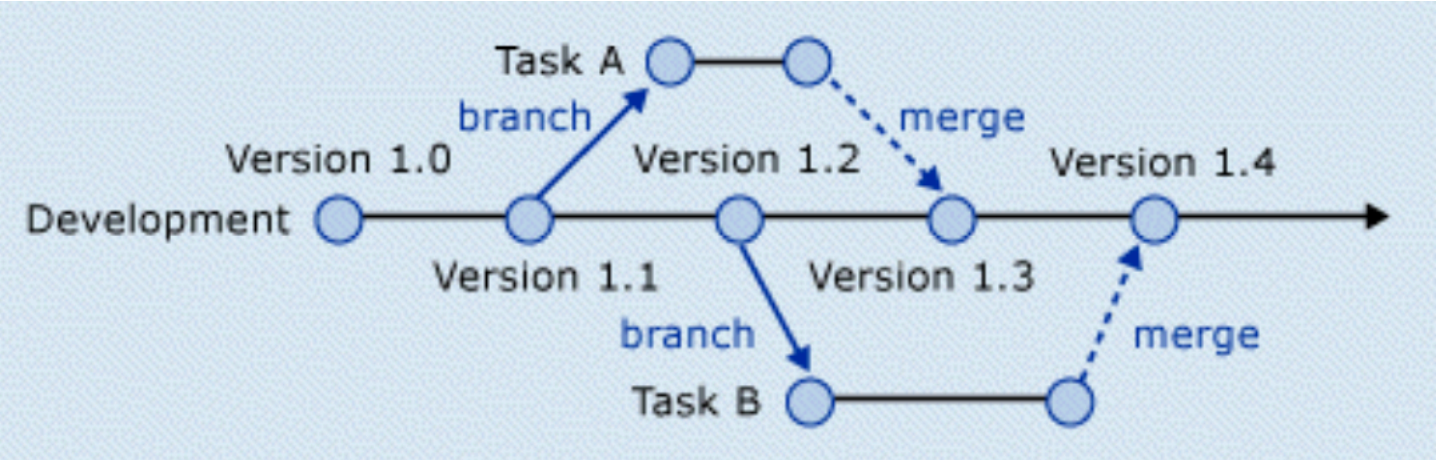
# Estrategia de Ramificación por Liberación



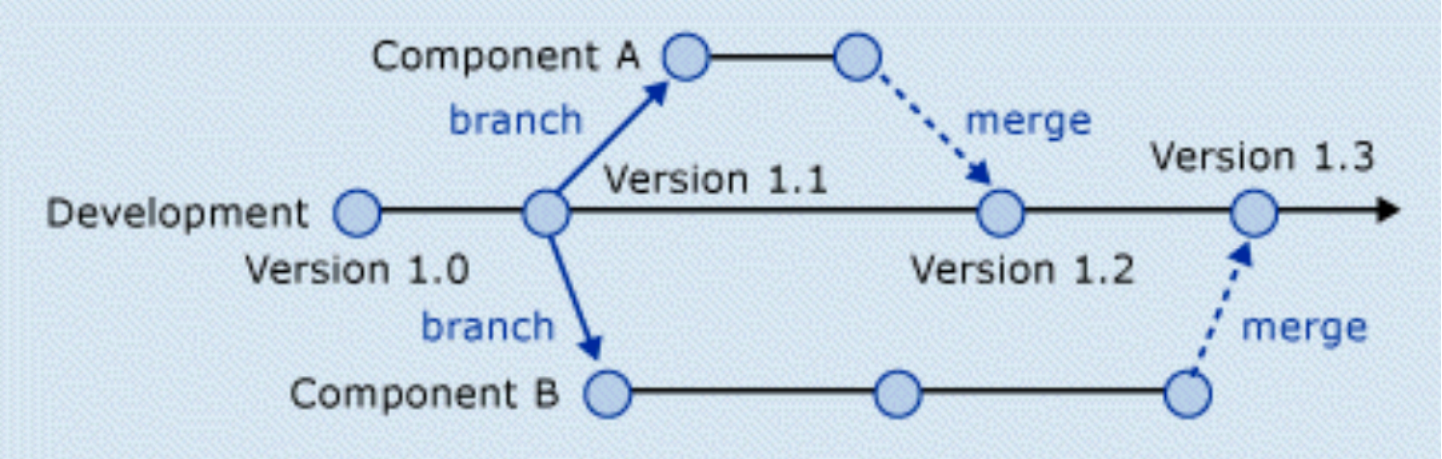
# *Estrategia de Ramificación por Promoción*



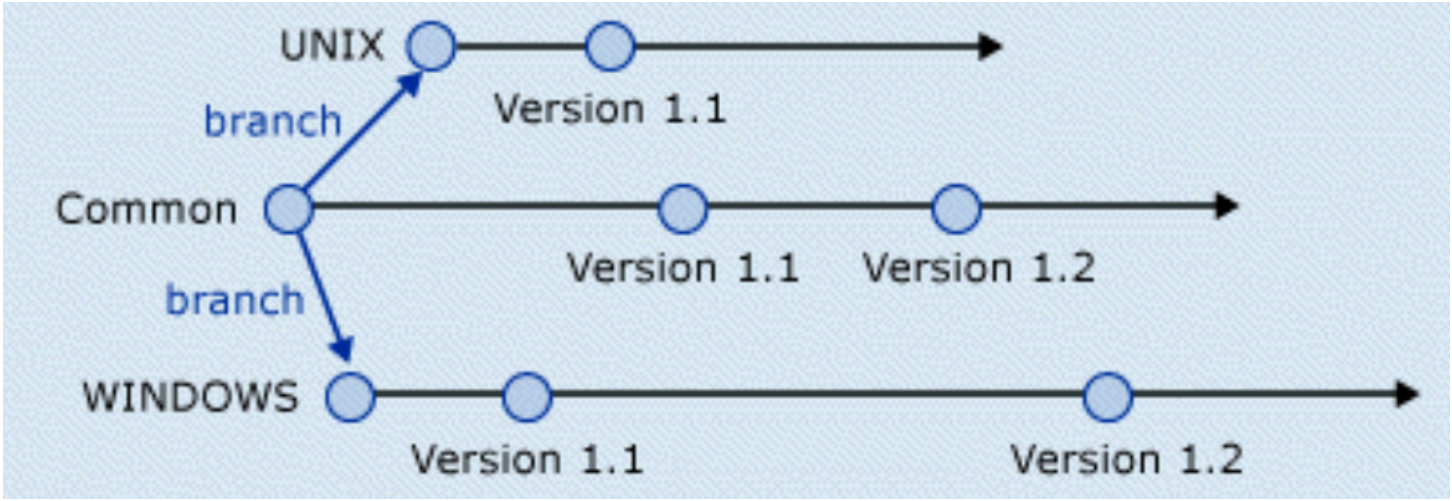
# Estrategia de Ramificación por Tarea



# *Estrategia de Ramificación por Componente*



# Estrategia de Ramificación por



# *Antipatrones*

- Merge-Paranoia
- Merge-Mania
- Big Bang Merge
- Never-Ending Merge
- Wrong-Way Merge
- Branch Mania
- Cascading Branches
- Development Freeze
- Berlin Wall

# GitHub

- GitHub es una plataforma de desarrollo colaborativo de software para alojar proyectos utilizando el sistema de control de versiones Git. El código se almacena de forma pública, aunque también se puede hacer de forma privada, creando una cuenta de pago.

# GitHub(2)

| `git init` |
| --- |
| initialize an existing directory as a Git repository |
| **`git clone [url]`** |
| retrieve an entire repository from a hosted location via URL |

# GitHub(3)

| |
|---|
| **`git remote add [alias] [url]`** |
| add a git URL as an alias |
| **`git fetch [alias]`** |
| fetch down all the branches from that Git remote |
| **`git merge [alias]/[branch]`** |
| merge a remote branch into your current branch to bring it up to date |
| **`git push [alias] [branch]`** |
| Transmit local branch commits to the remote repository branch |
| **`git pull`** |
| fetch and merge any commits from the tracking remote branch |

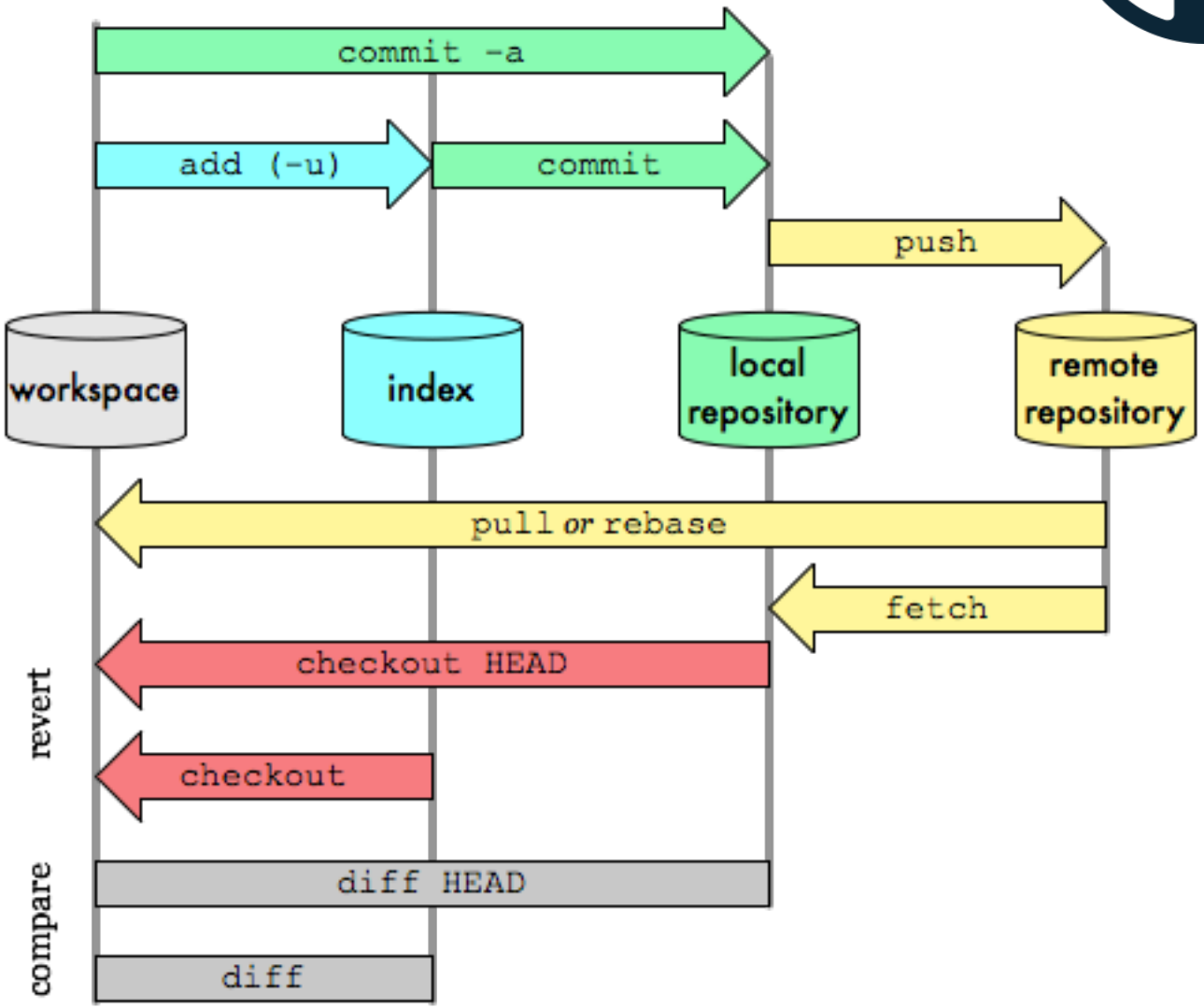
# GitHub(4)

| |
|---|
| **`git status`** |
| show modified files in working directory, staged for your next commit |
| **`git add [file]`** |
| add a file as it looks now to your next commit (stage) |
| **`git reset [file]`** |
| unstage a file while retaining the changes in working directory |
| **`git diff`** |
| diff of what is changed but not staged |
| **`git diff --staged`** |
| diff of what is staged but not yet committed |
| **`git commit -m "[descriptive message]"`** |
| commit your staged content as a new commit snapshot |

# GitHub(5)

| |
|---|
| **`git branch`** |
| list your branches. a * will appear next to the currently active branch |
| **`git branch [branch-name]`** |
| create a new branch at the current commit |
| **`git checkout`** |
| switch to another branch and check it out into your working directory |
| **`git merge [branch]`** |
| merge the specified branch's history into the current one |
| **`git log`** |
| show all commits in the current branch's history |

# GitHub(7)


commit -a
add (-u)
commit
push
workspace
index
local repository
remote repository
pull or rebase
fetch
checkout HEAD
revert
checkout
compare
diff HEAD
diff

# GitHub (8)

**https://try.github.io/levels/1/challenges/1**