SCM

Source Control Management

Powered by ‹epam›

CLOUD & DEVOPS COLOMBIA

CONTENT

- What is a SCM?
- History
- SMC Types
- Terminology

- Git
- Git structure
- Git States
- Basic Commands

- Branching
- Pull Request
- Undoing changes
- Conflicts
- Tagging

CLOUD
& DEVOPS
COLOMBIA

Ivan Gonzalez

Systems Engineer | DevOps Engineer

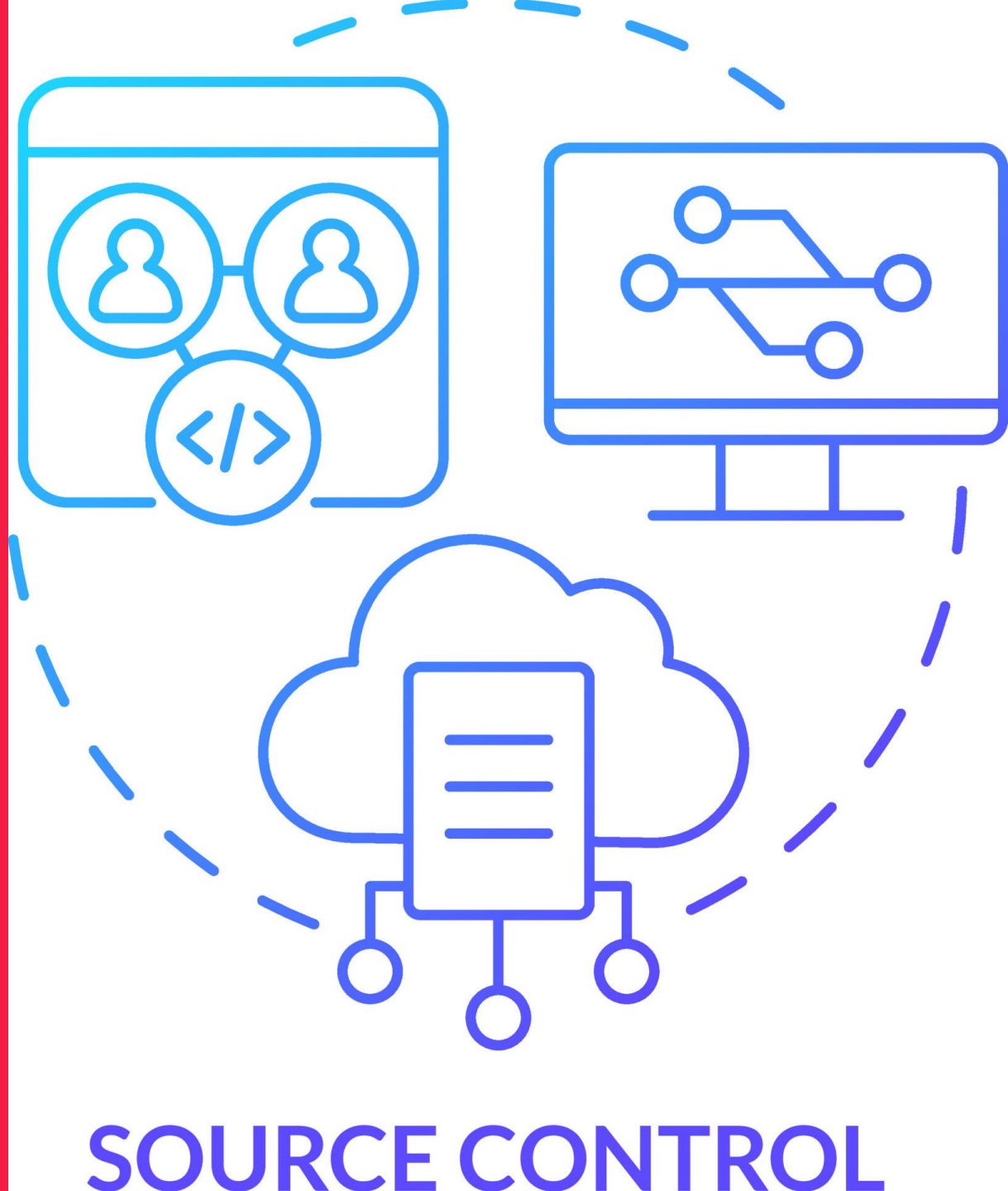
With 4+ years of experience in DevOps, I manage to get enough experience to share with people that are just beginning their journey into DevOps and its world.

1. Some context

What is SCM?

Change management over source code, assets, configuration files, etc., where modifications can be tracked.

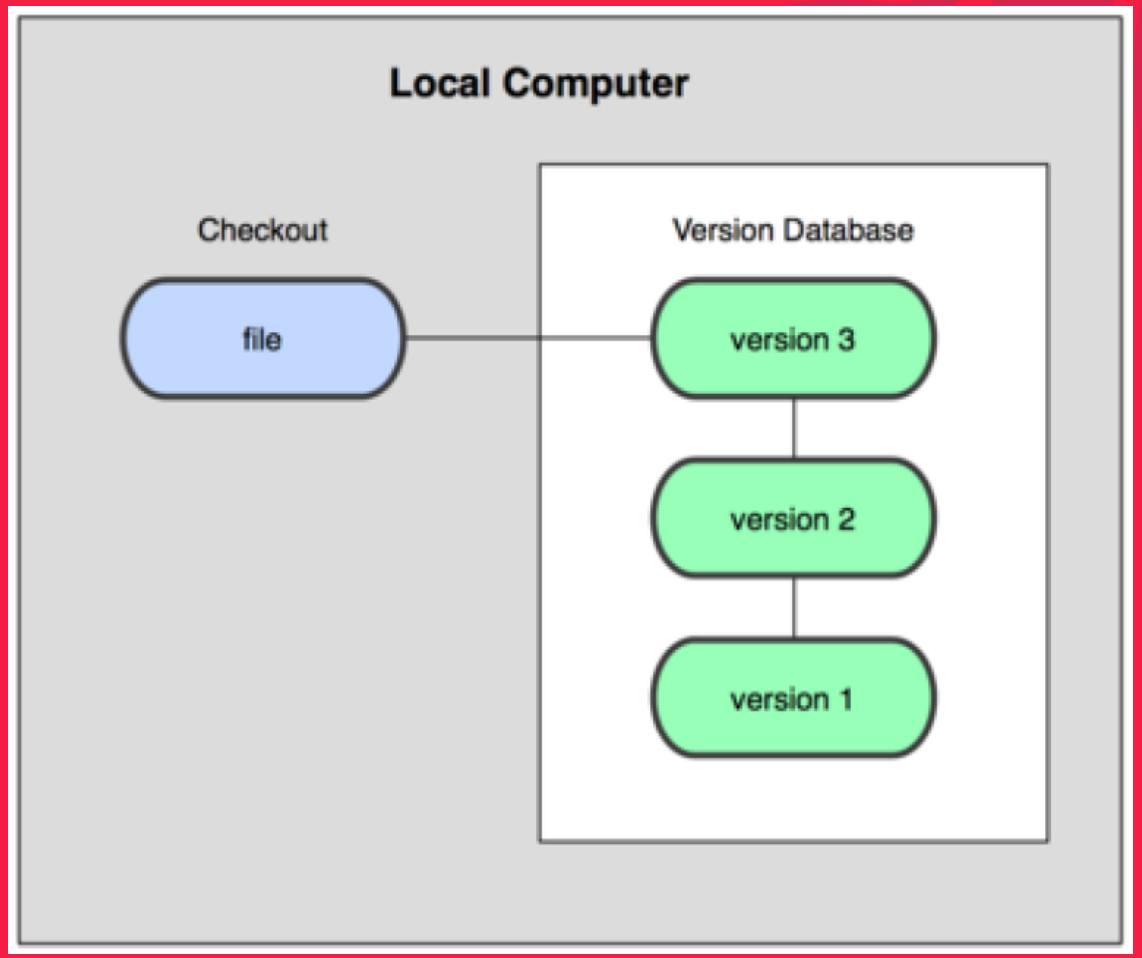
Can manage change logs, preventing accidents and losing information in the worst case secenario.

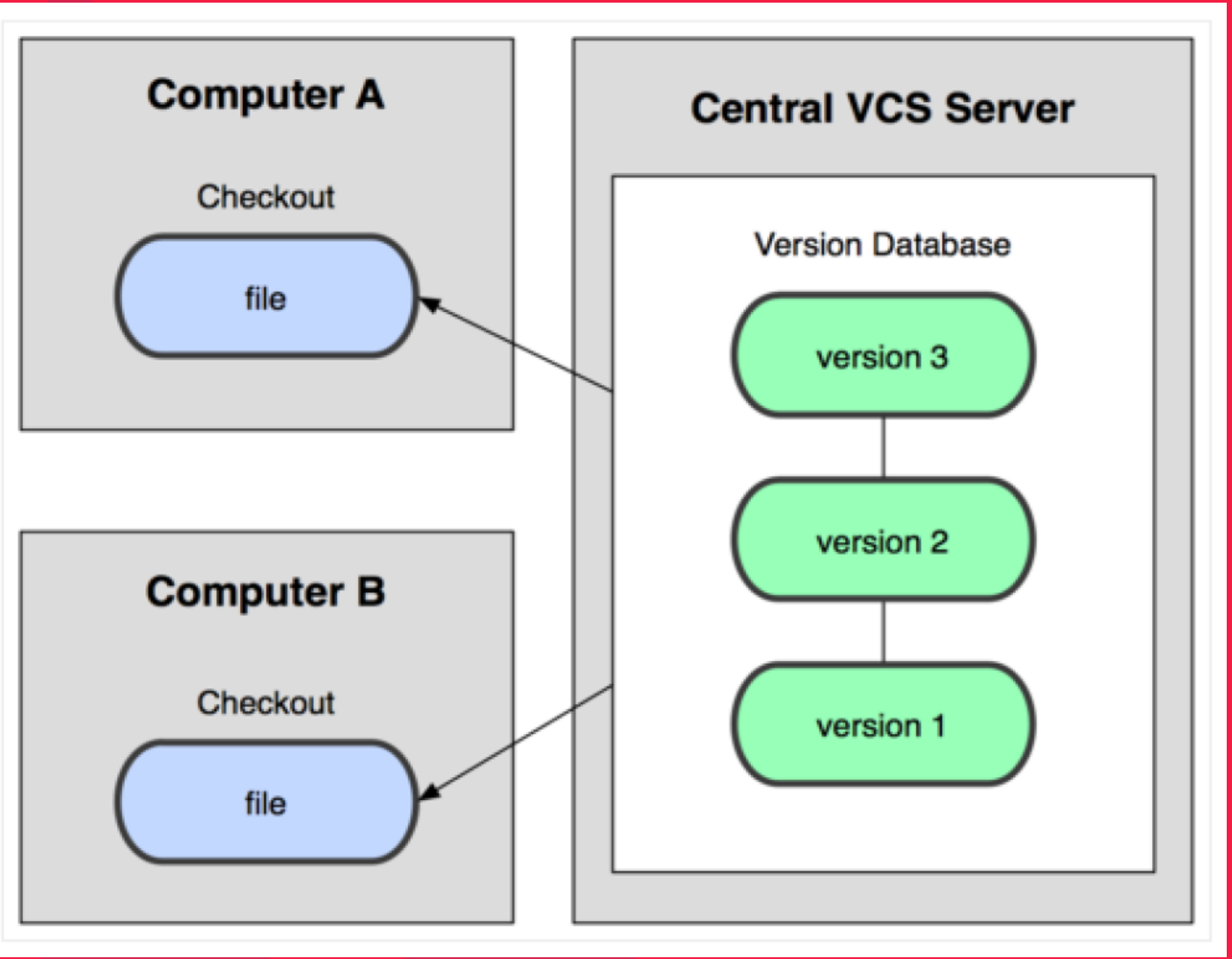


History

- Isolated and focus on files
  - SCCS, 1972. UNIX exclusive.
  - RCS, 1982. Multiplatform, but only text.
- Centralized
  - CVS, 1986. First centralized repository, focusing on files.
  - Perforce, 1995. Used by Google.
  - Subversion, 2000. Works for executable files, and use folder for its basic unit.
  - Microsoft Team Foundation Server, 2010. Comes with MSDN susbscribtion, being a previous version of Azure DevOps.
- Distributed
  - Git, 2005. Created by Linus Torvalds after BitKeeper going commercial. Is the standard in the industry.
  - Mercurial, 2005. Also created to answer BitKeeper change.

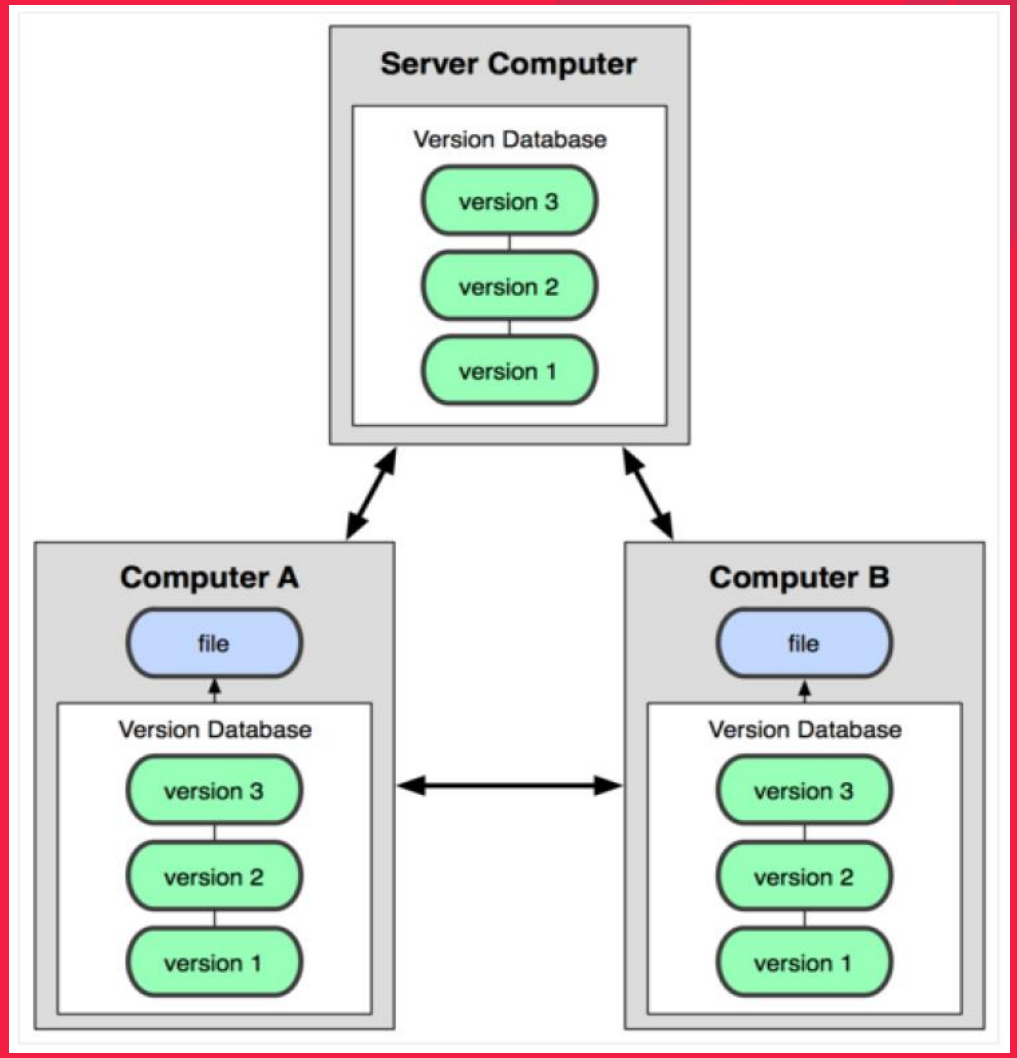
SCM Types (Local)





SCM Types (Centralized)

SCM Types (Distributed)



Terminology

Repository:

Place where the changes are saved.

Change:

Difference in a file from a previous version to the current version.

Commit:

Confirm the changes made in files in a repository.

Revision:

Unique identifier of a change set. For example: git hash.

Tag:

Code importart revisions, mostly, productive versions.

Trunk:
Repository base line.

Head:
Last revision made in a trunk.

Branch:
Forks of a trunk, where different changes are generated with different heads.

Merge:

Join between several banches.

Conflict:

Differences found on files that can be resolved during branch merge.

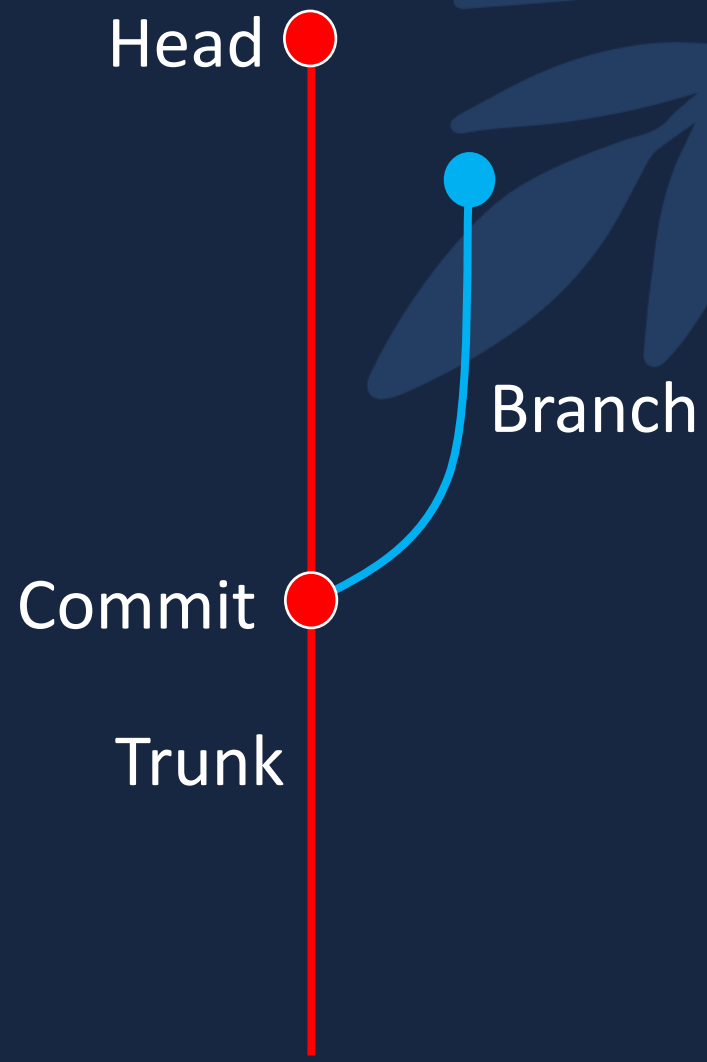
2. Git basics

Git

- Distributed SCM.
- Better performance against other systems due to algorithm optimization.
- Secured using SHA-1 encryption.
- Can be used in projects of all sizes.
- Industry standard.
- Open source.

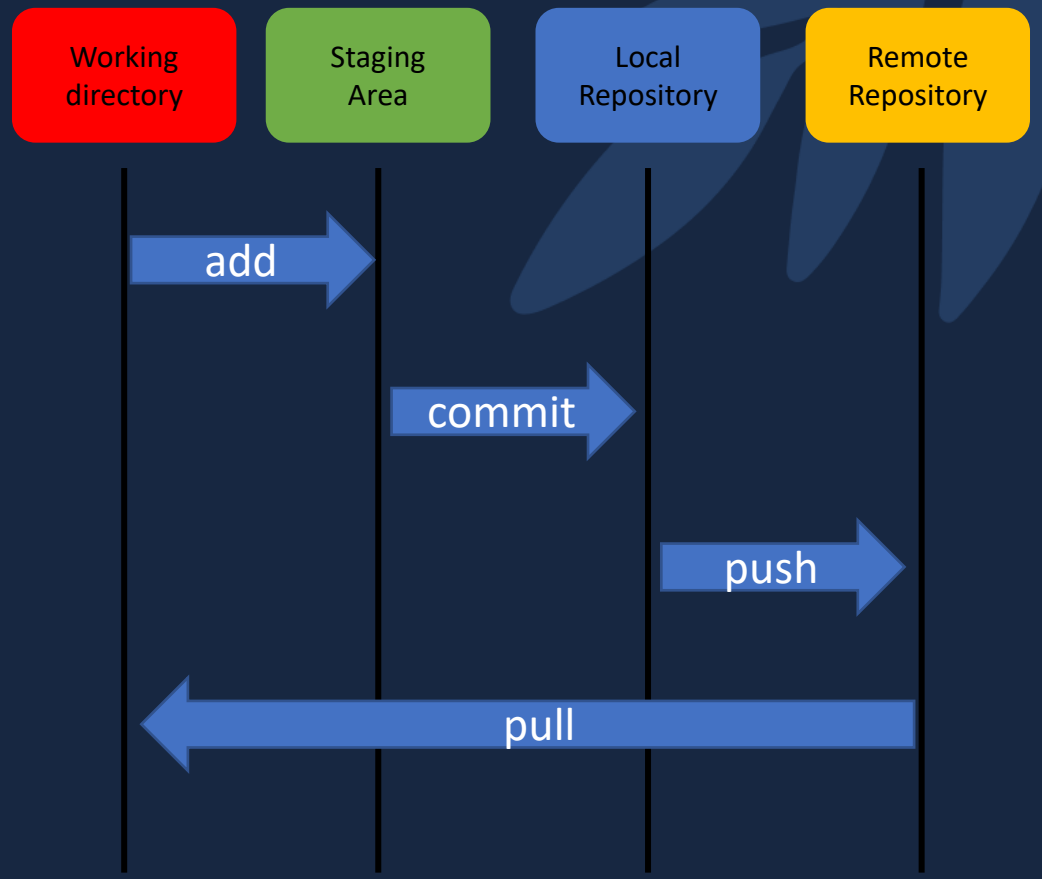
Git structure

- Trunk: Repository base line.
- Commit: Confirm the changes made in files in a repository.
- Head: Last revision made in a trunk.
- Branch: Forks of a trunk, where different changes are generated with different heads.



Git states

- Local folder: current workspace.
- Staging area (ready to commit): where the files are ready to be committed.
- Local repository: The files are versioned locally.
- Remote repository: Final state where the files are versioned on a remote location (usually the SCM preferred tool).



Basic Commands

- Git init: Start local repository.
- Git remote: Manage remote repositories.
- Git add: Add files to Staging Area.
- Git status: Staging area and folder state.

```
git init

git remote add name_of_the_remote(e.j:origin) remote_url

git remote rename old_name_of_the_remote new_name_of_the_remote

git remote remove name_of_the_remote(e.j:origin)

git add [name_files_or_dot(.)_for_all_files_in_folder_with_changes]

git status
```

Basic Commands

- Git commit: Confirm the changes in the staging area and versioning locally.
- Git push: Send the changes from the local repository to the remote repository.
- Git pull: Download the changes from the remote repository to the local repository.

```
git commmit --message "Commit message"

git push

git pull
```

Basic Commands

- Git fetch: Download a snapshot of the changes which are in the remote repository to the local repository.
- Git log: Allow to look at commit logs made in a branch or trunk.
- Git clone: Download a remote repository.

```
git fetch

git log

git clone url_of_the_repo
```

3. Middle/Advance concepts

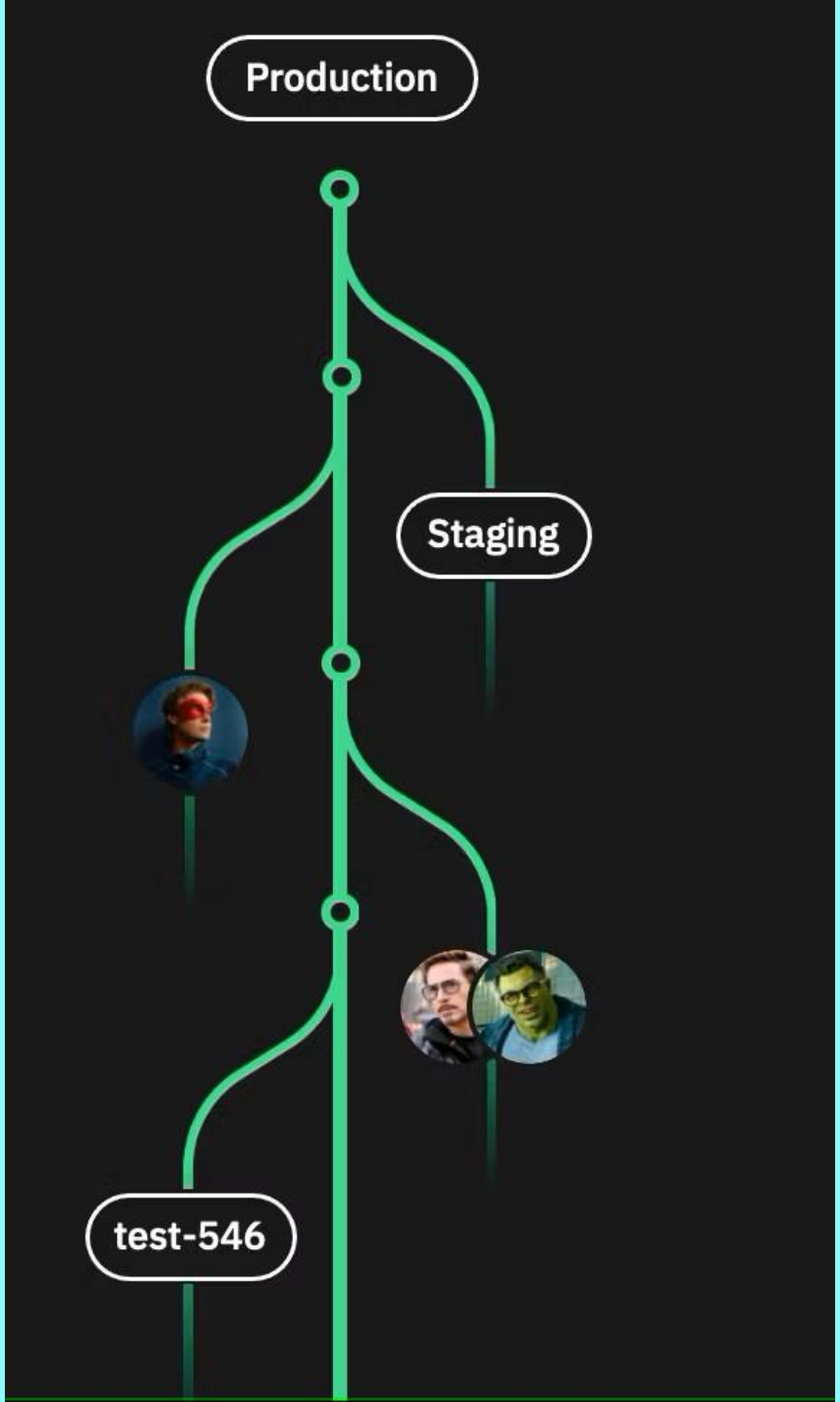
Branching

Workflow creation in a repository, where forking from the main trunk, allows to work in a isolated environment without changing the parent trunk.

Allows collaborative workflows, without generating dependencies of affects the work of other colleagues.

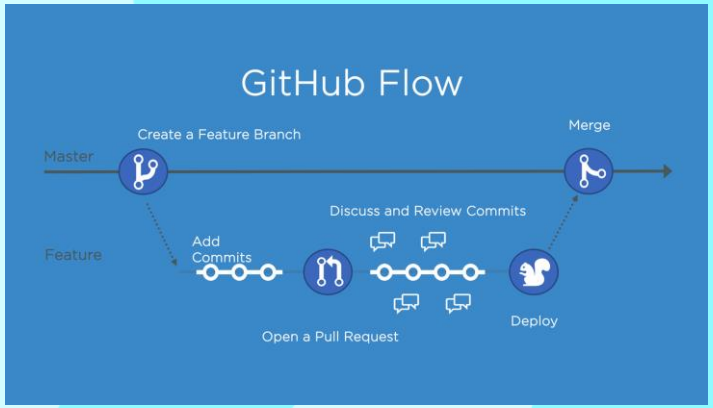
Allows several configurations for environments.

Make easy to work on complex and big projects.



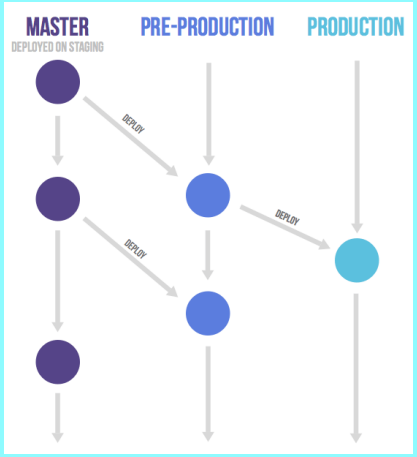
Branching: Flows

To manage braching, there are flows that can be used according to the project needs. This are some examples:



GitHub Flow:

Light branching which only use Master and Feature branches, where the last one is for working to tasks and Master is the production branch.



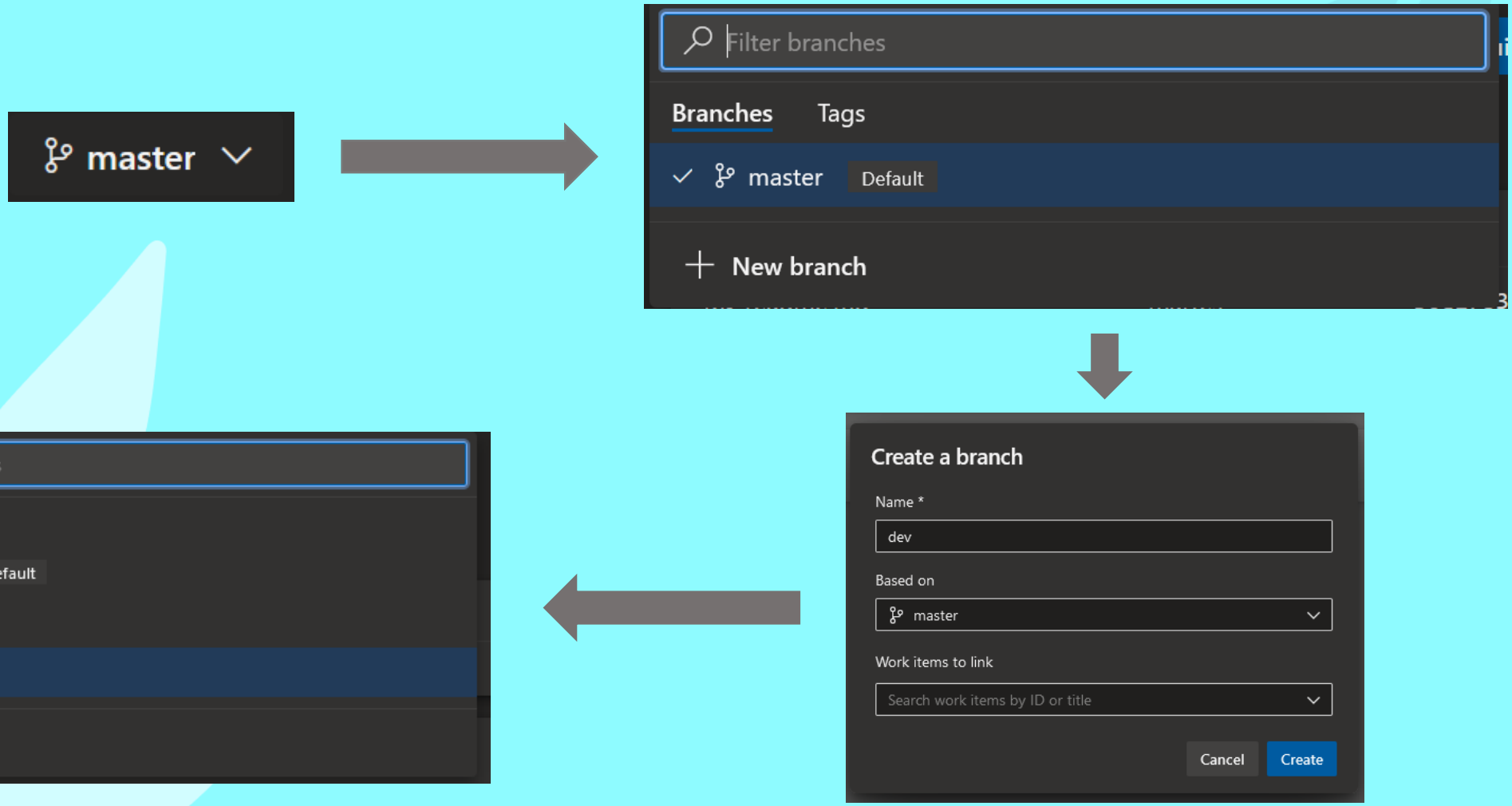
GitLab Flow:

Branching focus on having a Branch per environment.

Git Flow:

Branching with 2 main branches, master & dev, where from dev branch into feature & release branches, which is the branch to deploy into master.

Branching: create remote branchs



Branching: create local branchs

Git Branch: List, create and delete branches created locally and remotelly.

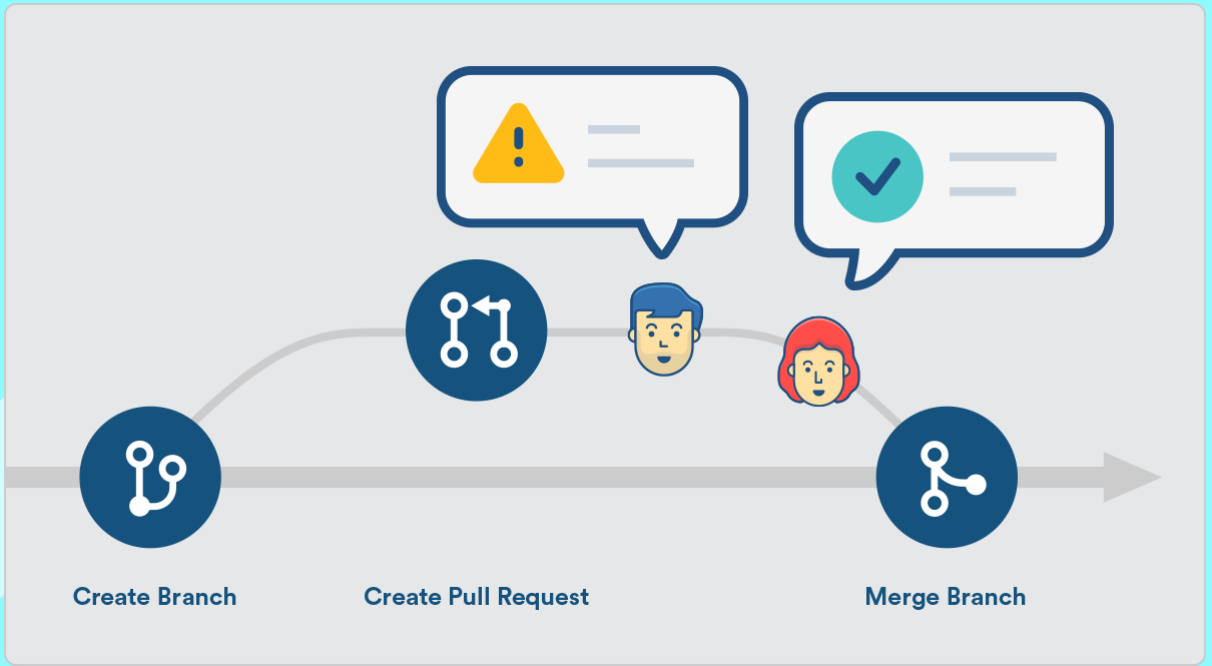
`git branch`	List local branches	`git branch [name_of_the_branch]`	create local branch
`git branch -a`	List remote branches	`git branch -d [name_of_the_branch]`	delete local branch

Git checkout: Allows moving between local branches/commits

```
git checkout [name_of_the_branch | commit_hash]
```

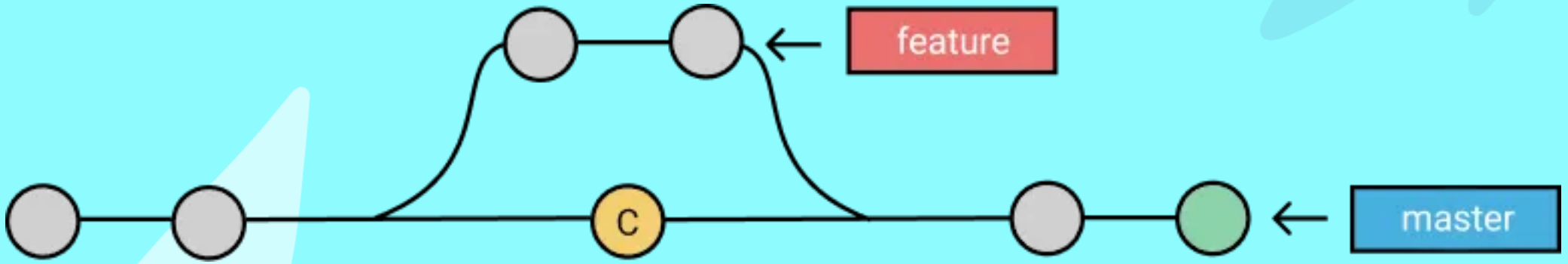
Pull Request

- Integration between branches.
- Allows pair revisions.
- Allows authorization to allows merges to certain branches.
- Allows comments to the revisions.
- There are cases where pull request can stay open and be close.



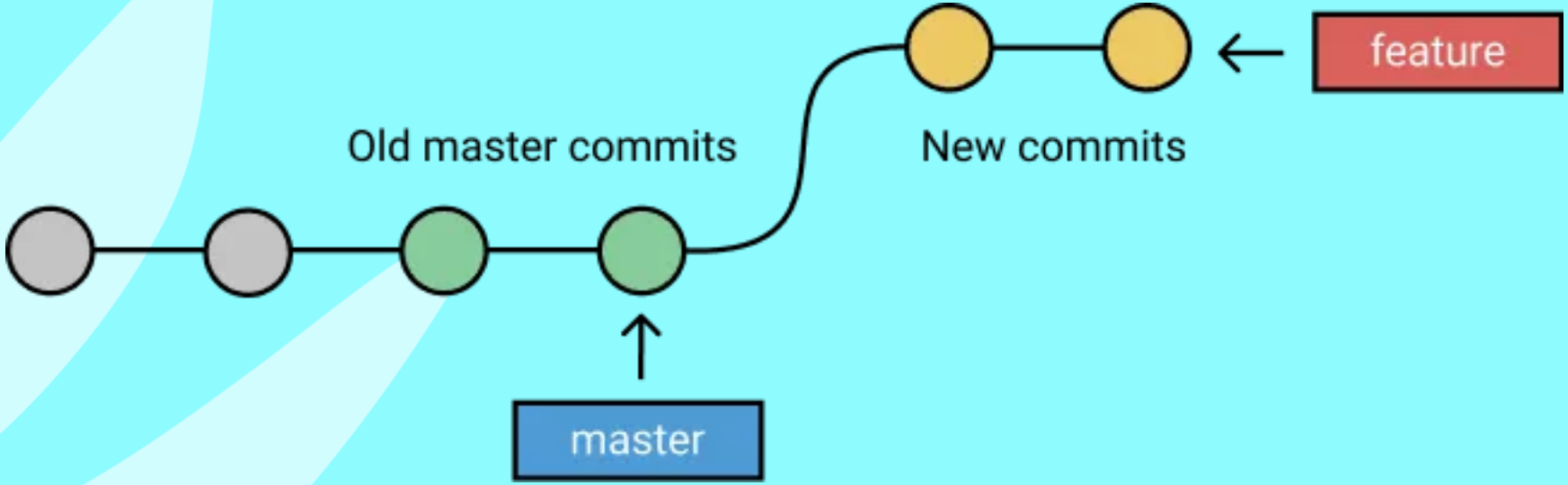
Pull Request: Merge

- Preserve commits history
- Recommented for big developments

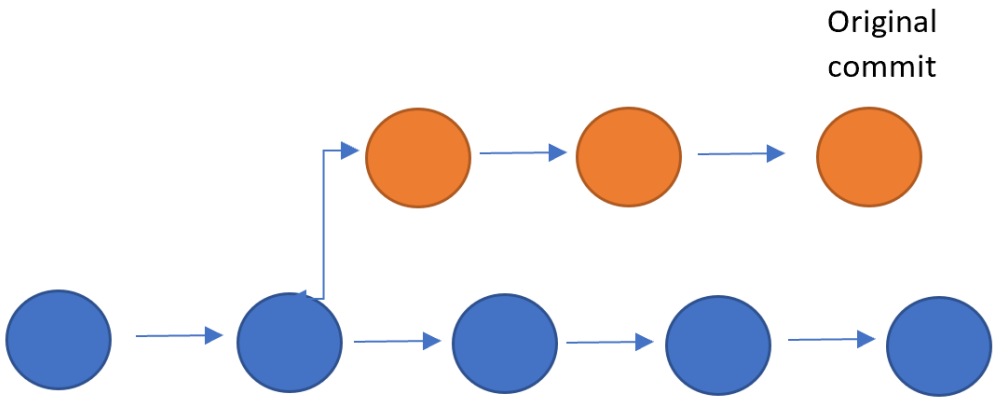


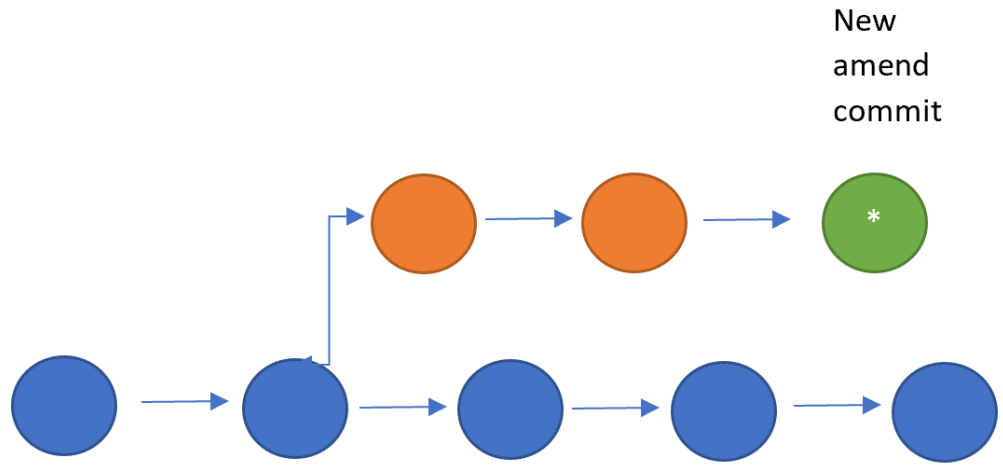
Pull Request: Rebase

- Doesn’t preserve commits history.
- Usefull in small repositories where history is not important.
- Usefull when feature branches history is not relevant or can be messy to read.
- Can have an interactive rebase.



3.1. Undoing Changes





Amend commits

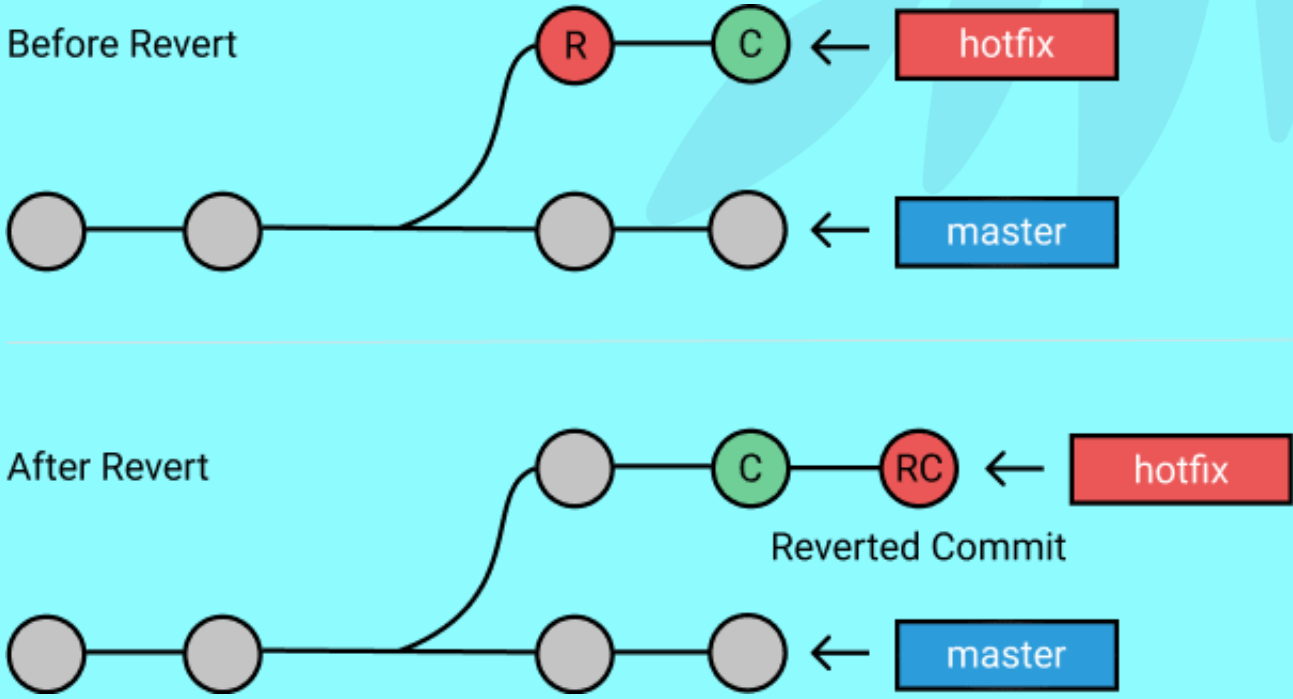
- Edit the last commit before it goes to the remote repository, changing files and the message.
- To change the message:
- git commit --amend:
- With only this command, there is an interactive text editor where you can change the message.
- With -m flag, you can write the message for the commit (ej: git commit --amend -m “new message”).
- If you want to leave the commit message unchanged, you can use the --no-edit flag.
- To add or remove files:
- git add “file_to_add” or git rm “file_to_remove”
- Then, git commit --amend.

Undoing changes

- git rm:
  - Remove tracked files from the git's staging area, undoing the changes on those files.
- git clean:
  - Clean untracked changes (changes not added into staged state).
- git checkout:
  - Allows to move a certain commit into a new branch, to reset the branch into that commit using a merge or a pull request.

Git revert

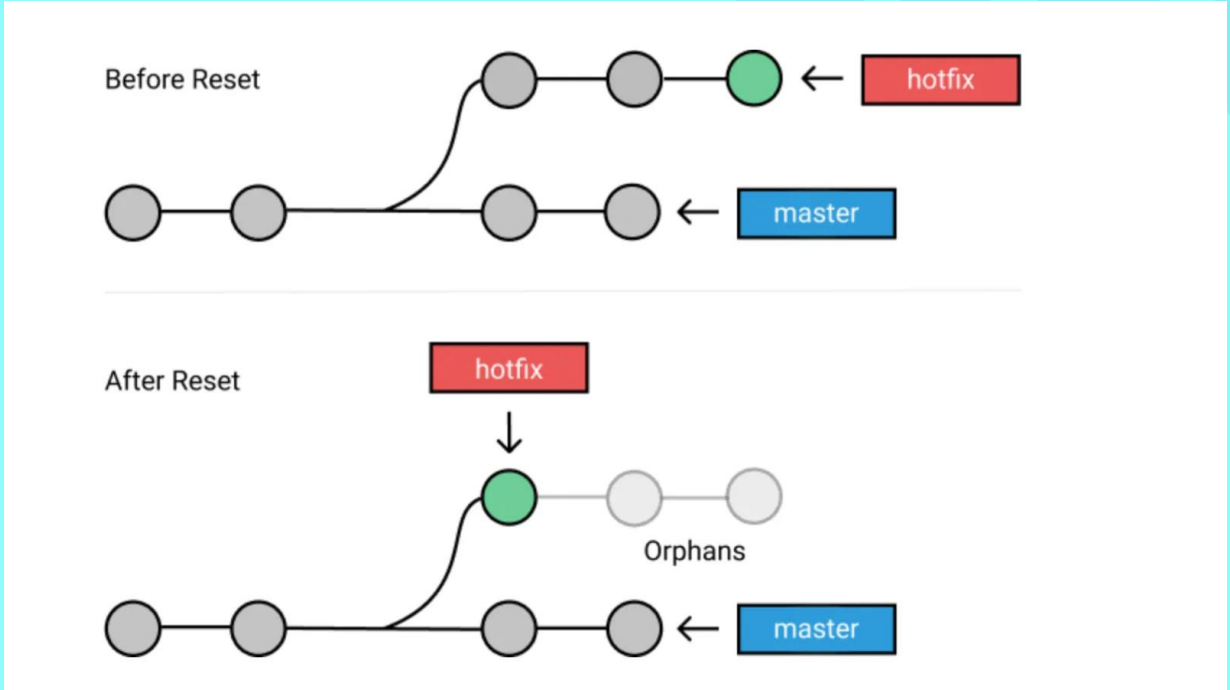
- Safely return the head of a branch to a known state by creating a new commit with the state of the desired commit.

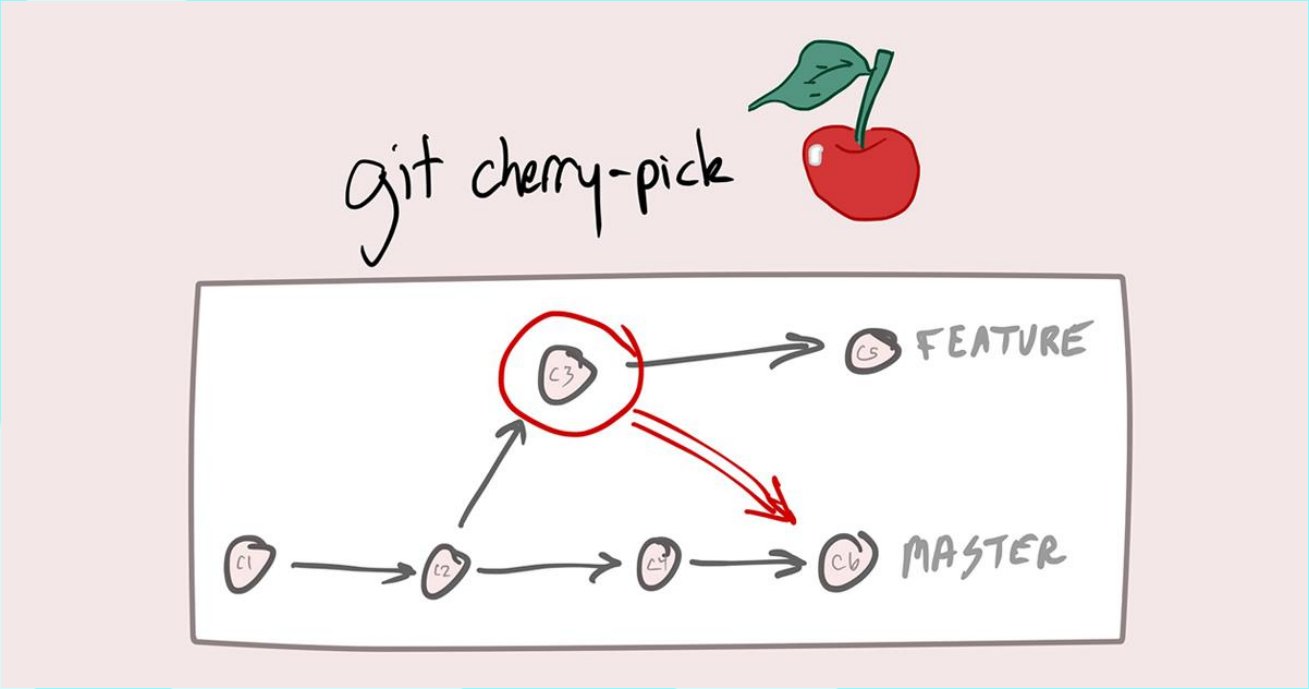


Git reset

Move the branch's head into a desired commit, resetting the head to that commit. It can be --hard, --soft or --mixed.

- If is --soft, the reset doesn't reset the working directory and the staging area, only the commit reference is reset.
- If is --mixed, the staging area is reset and the changes in that area, go back to the working directory. This is the default mode.
- If is --hard, the staging area and the working directory are reset. This is the most dangerous mode.





Cherrypicking

- Allows to pick a commit and move it in the same branch or to another branch.
- Is another way to undo changes.
- Is good for team collaboration, when there are frontend and backend are working in the same feature, using a certain commit to build frontend.
- It can have some application to bug hotfixing, where a bug is identified and fixed, that commit can be cherry-picked into master/main branch.

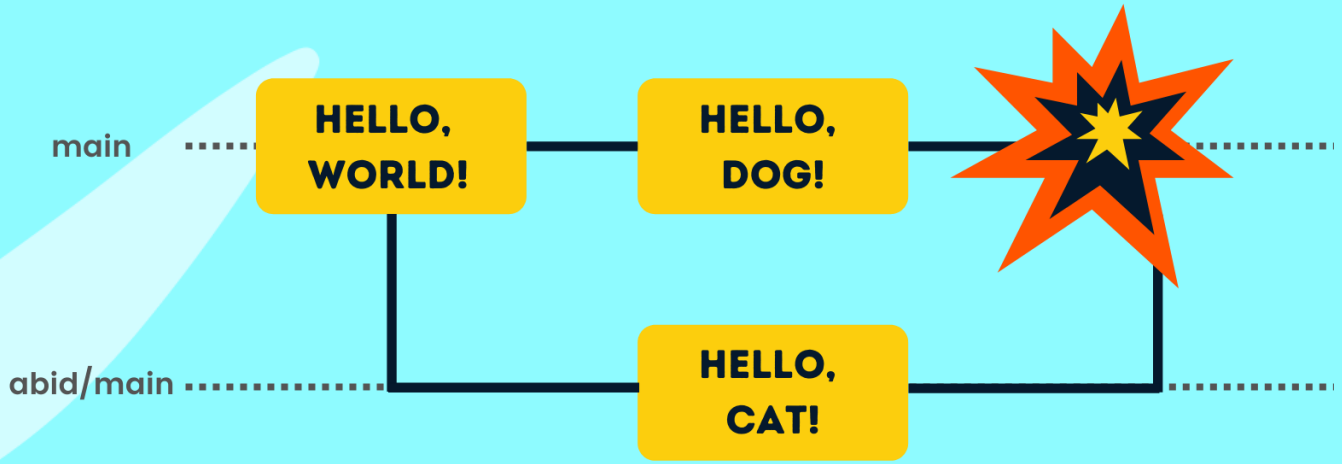
End of 3.1. Undoing Changes

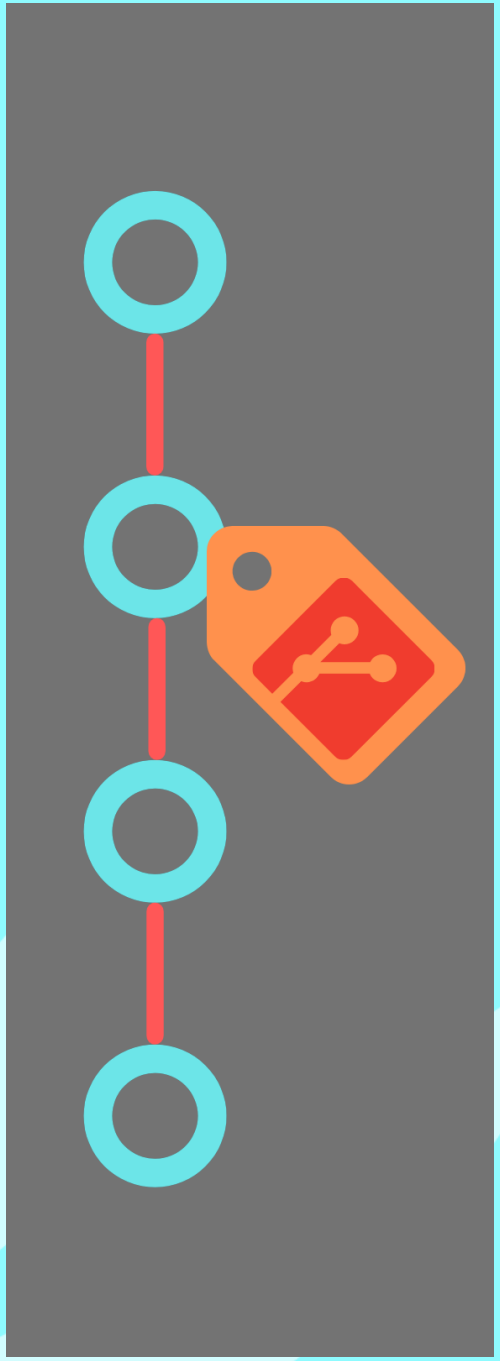
Conflicts

- A conflict occurs when there are changes that overwrites the files and git is not available to choose which change are correct.
- This issue can be resolved by:
- Reading the conflict and check which files are affected.
- Modifying the file(s), resolving the changes and push into the branch.

This are some commands that may help to fix the conflicts:

- git log --merge: shows the commits that are causing the conflicts.
- git diff: helps to identify differences in the states of repositories/files.
- git checkout: reset file to its original before commit.
- git reset: reset files to a desired and known state (commit).





Tagging

- Version identifier for important releases, and mostly, to identify production ready code from master branches.
- Is like a branch that can’t be changed.
- Most of the naming notation for tagging is vMAJOR.MINOR.PATCH
- The command is: git tag -a version -m “Message”

THANK YOU!

Do you have any question?

ivan_Gonzalez@epam.com