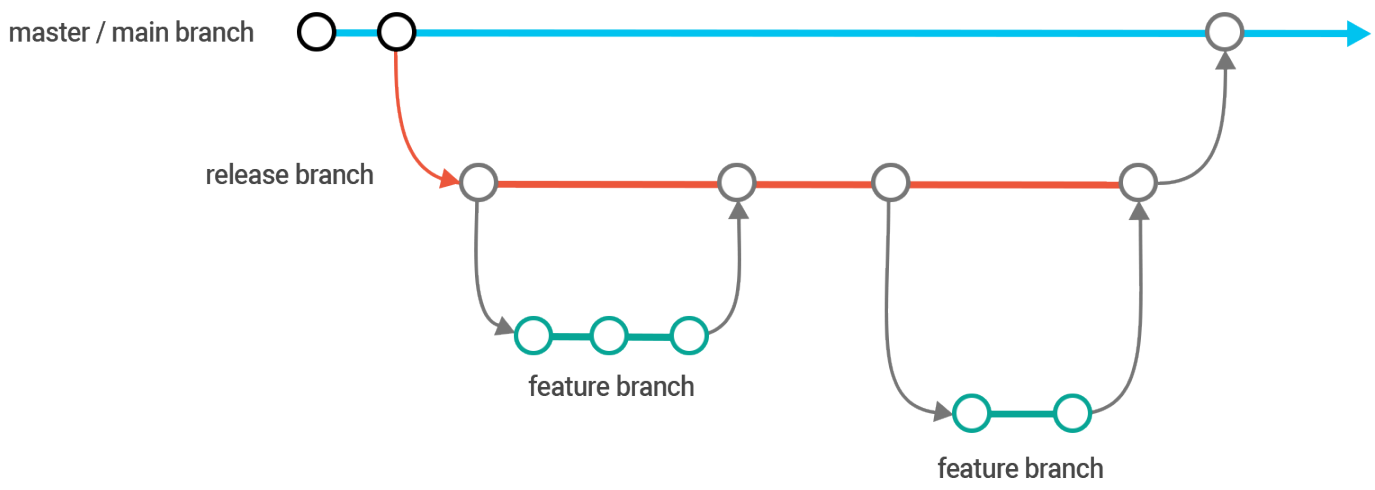
## VERSION CONTROL WITH GIT, THE SINGLE SOURCE OF TRUTH

A **version control system** tracks changes over time in a file or a set of files stored in a specific location, known as a repository. Version control with Git gives you the possibility of cloning the repository locally. This way, developers can have a copy of all the files in the repository at any time in their local machine. Let's take a look at the main elements of version control:

- **Files:** All developments can be represented as files in a specific format. Version control keeps the definition of a component in the source language/format.
- **Git Commit:** It records a change. A commit indicates that the content of a file has been modified.
- **Git Branch:** Commits are organized into branches. A branch collects the versions of the files and the commit history, which references the changes in these files over time.
  Branches can be created out of other branches and fed with contributions from other branches via a merge. You can create as many branches as you need in your release cycle, but the first branch to be created must be master or main.
- **Git Merge:** A Git merge integrates changes from one branch into another and keeps your branches aligned.
- **Git Repository:** It is the place where file versions, branches and commits are stored. There are two types of repositories:
- **Remote (On Cloud or hosted on a server).** The version of the repository is accessed through the browser.
- **Local.** A copy of your repository in a local machine (computer).

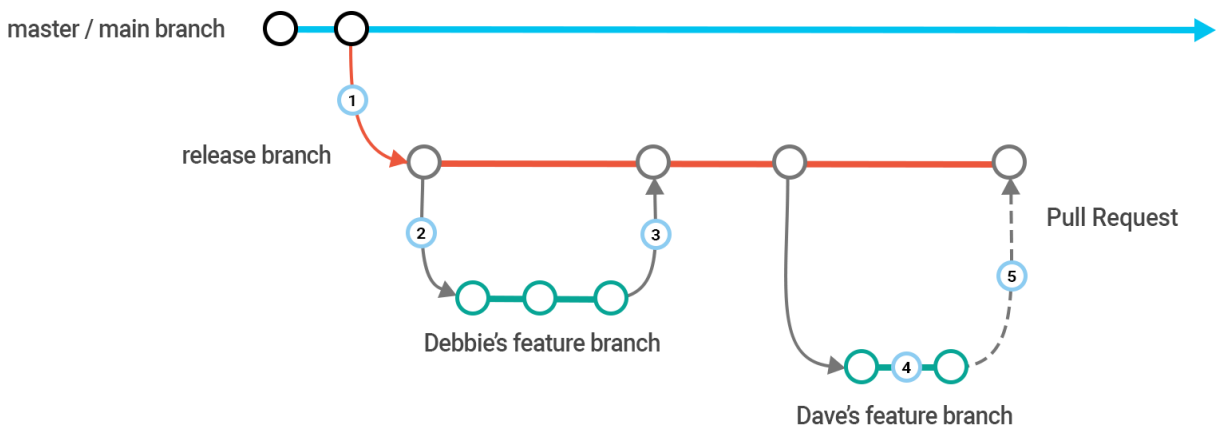
## KEY CONCEPTS OF TRADITIONAL BRANCHING MODEL




- **Master/Main Branch:** This branch contains the production-ready version of your code delivered to your users.

- **Release Branch:** It is branched off from master and groups the features branches' changes you have been working on. After all the developers have integrated their contributions in the release branch, the release branch is merged and deployed into the master branch.
- **Feature Branch:** It is generated from the release branch. This is the branch where developers apply the modifications related to the features. Once a developer is done with the changes, the feature branches are merged into the release branch.
- **Pull Request:** A pull request is essentially an approval process before merging two branches so that the changes are introduced in a controlled way.

## TRADITIONAL BRANCHING MODEL PROCESS



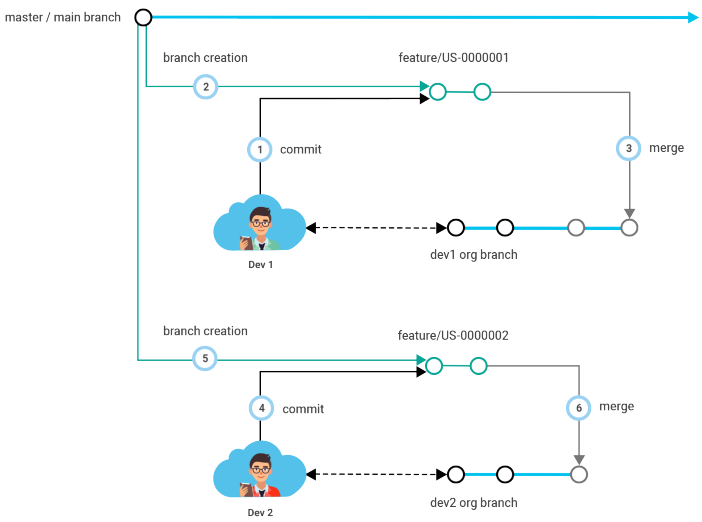

**1.** The first step the team should take is to generate a new release branch out of master. A developer (Debbie) creates a local feature branch out of the release branch and starts committing her work.
**2.** Then she creates a pull request against the release branch so that other teammates can review her work. After the pull request approval, Debbie merges the feature branch into the release branch to push the changes.
**3.** A new developer, Dave, is taking care of other release requirements, so he branches off another feature branch from the release branch to work locally. His local copy already incorporates Debbie's commits.
**4.** After committing his changes Dave performs a pull request against the release branch to verify that the feature branch content is mergeable.
**5.** The pull request gets approval from the team, enabling Dave to merge his feature branch into the release branch.

## COPADO BRANCHING MODEL

| Branch | Description | Name |
|---|---|---|
| Master / Main branch | It represents your Production Org. | master/main |
| Feature branch | An isolated branch that contains your changes. | feature/{UserStoryName} |
| Promotion branch | An isolated branch created from the destination branch. It is an intermediate branch that drives the deployment to the next or previous org in your flow. | promotion/{PromotionName}-{DeploymentName} |
| Org branch | Branch linked to the environment where you have applied the changes. | dev1, dev2, uat, stage, master/main |
| Destination branch | Branch linked to the org where you would like to deploy your changes. | dev1, dev2, uat, stage, master/main |

## COMMIT PROCESS

In the Copado model, each Salesforce environment is tied to a branch in the repository. In between each branch representing the different environments, Copado makes use of feature branches and promotion branches. Let's see an example:



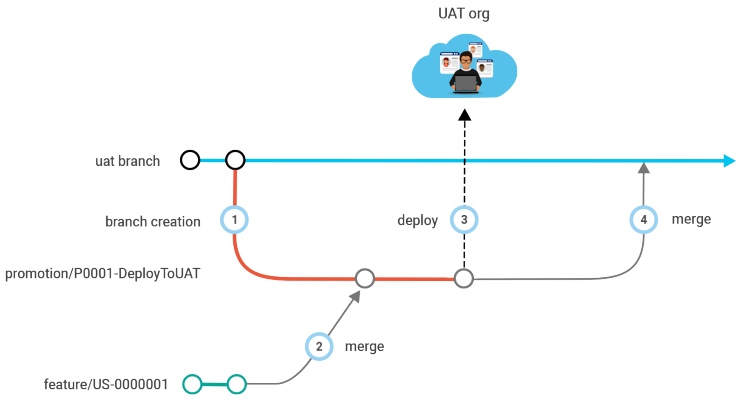

**1.** When Dave commits in his user story (1) Copado creates a **feature branch** (2) that captures all the commits related to his user story.

By default, feature branches are generated from the main branch of the pipeline, usually master. This is because Production is considered a stable environment.

**2.** Before promoting the user story, he creates a pull request from the user story to compare the user story changes against UAT. The pull request is made between the user story feature branch and the destination branch to see how his changes will impact the destination Org.

**3.** Debbie has been working on her user story in Dev2. When she commits in her user story (4) Copado creates a new feature branch for her from master, keeping her changes separated from Dave's changes (5).

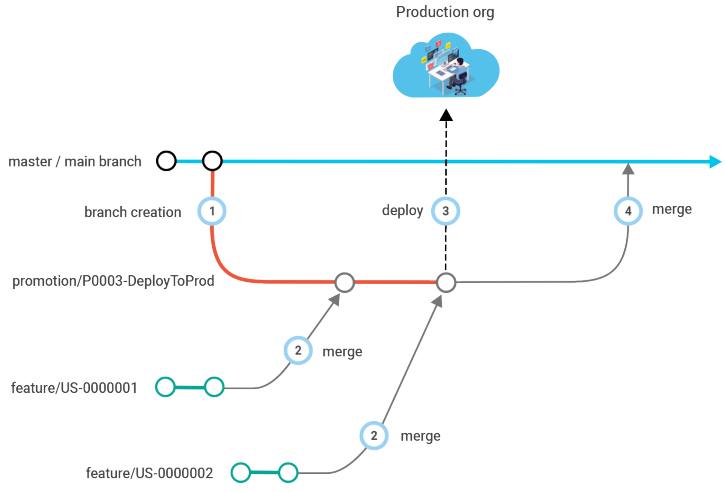
## PROMOTION TO UAT




**1.** When Dave's user story is promoted to UAT, a **promotion branch** is created out of the uat branch(1).
**2.** During this promotion process, if no conflicts are detected, the feature branch is merged into the Promotion branch (2). Thus, the original content of UAT remains, while the new contributions from Dev1 are added.

**3.** If multiple user stories are included in a Promotion record, their respective feature branches will be merged sequentially into the Promotion branch.
**4.** Dave executes the Deployment. The source used for the deployment will be the files as they are in the Promotion Branch at the time of the latest commit.
**5.** Once the Deployment has been successfully completed (3), the Promotion branch is merged into the uat branch (4), incorporating the changes committed in the user story into the destination Org branch.

## PROMOTION TO PRODUCTION




The user stories are ready to be deployed to Production.

**1.** The release manager includes both in a promotion and creates a deployment. As he creates the Deployment, a promotion branch is created out of the master branch (1).
**2.** The feature branches are reused and merged into this Promotion branch (2).

**3.** Let's imagine that the Business chooses not to move Debbie's user story in this release. With Copado, the release manager only needs to deselect Debbie's user story from the Promotion record and submit a new deployment. This will create another promotion branch without Debbie's feature branch.

**4.** Following a successful deployment (3), the new Promotion branch will get merged into master, and the user story commits will reach the final stage (4).

# BENEFITS OF WORKING WITH COPADO

|  **Isolated individual work** |  **Time-saving** |  **Traceability** |
|---|---|---|
| If a deployment fails you can simply remove the affected user story and retry creating a promotion without this feature branch. | Copado eliminates the manual work, automating the creation of the feature and promotion branches and the merge of the Promotion branch into the Destination branch. | Copado follows a naming convention for the branches it creates so that you can quickly locate them in your repository. |

Interested in learning more? Go to:
https://success.copado.com/

COPADO ACADEMY