MIRANTIS

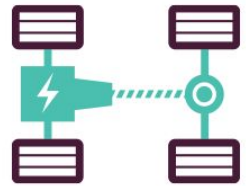
Git, Gerrit and Jenkins in MCP DriveTrain

Introduction

training.mirantis.com

Objectives

- What are: Git, Gerrit and Jenkins
- How they work together in MCP DriveTrain



DriveTrain

Git

Version Control System

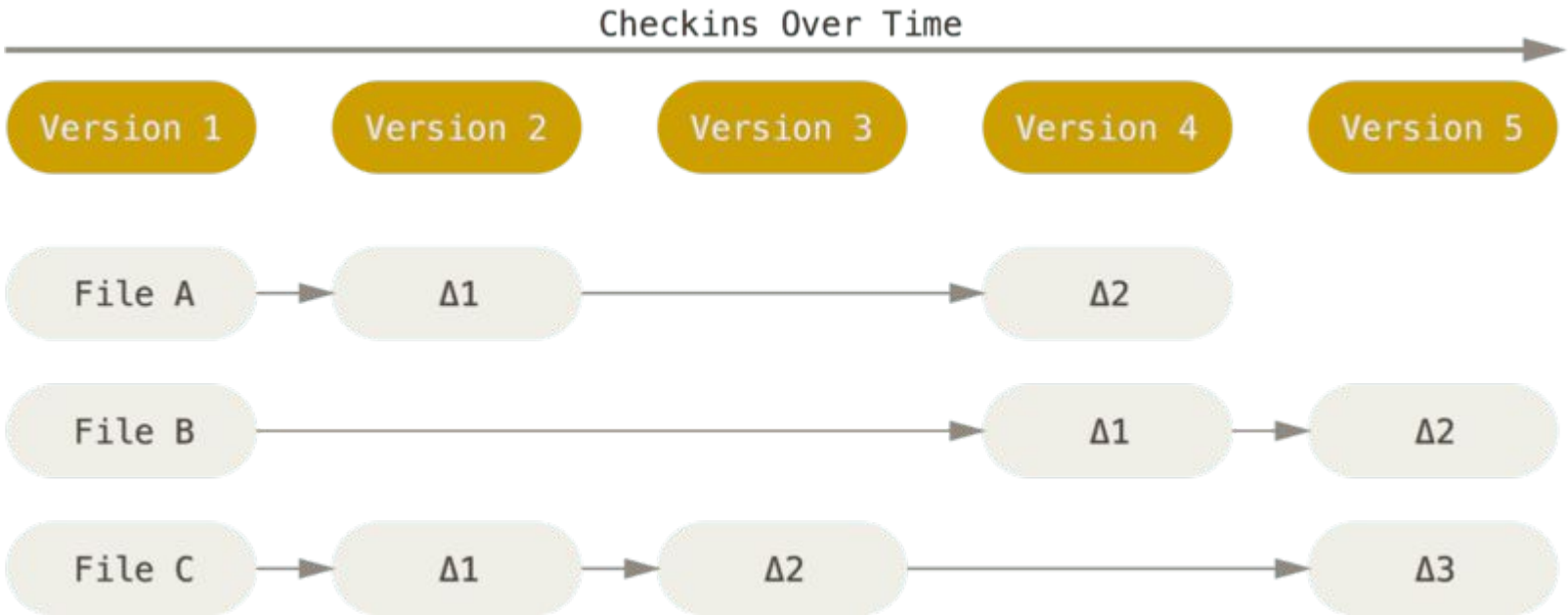
What is a Version Control System (VCS)?

- System to track changes to documents, code or any **collections of text files**

What is a Version Control System (VCS)?

- In the simplest form of a poor man's VCS
- you could store multiple copies of **different versions** of your collections
- in separate directories
  - It is not efficient, because not modified files are duplicated
- More efficient VCS could store a list of file based changes - **deltas**

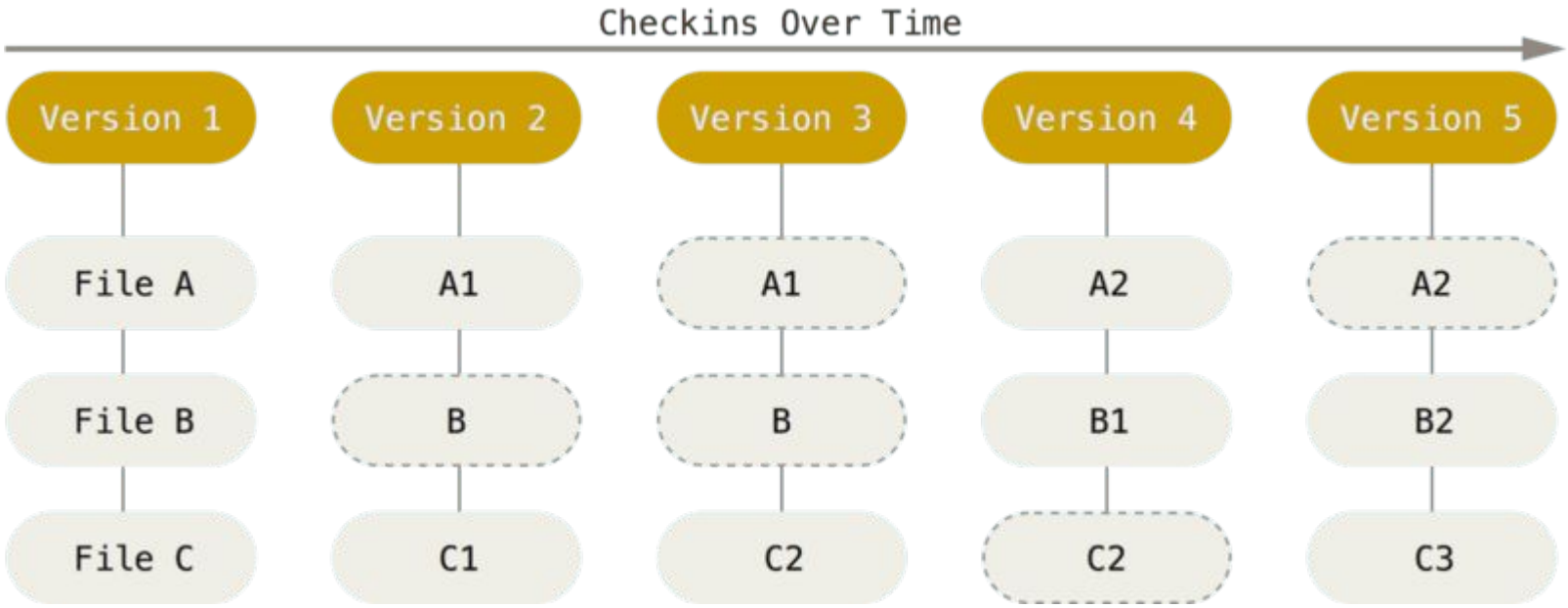
Delta based VCS



Snapshot based VCS

- Even better idea is to store versions as snapshots
- **Snapshot** is a picture of what all your files look like at that moment when changes were applied (**checked-in**)
- If a file has not changed, VCS stores just a link (reference) to the previous identical file

Snapshot based VCS



git --everything-is-local ™

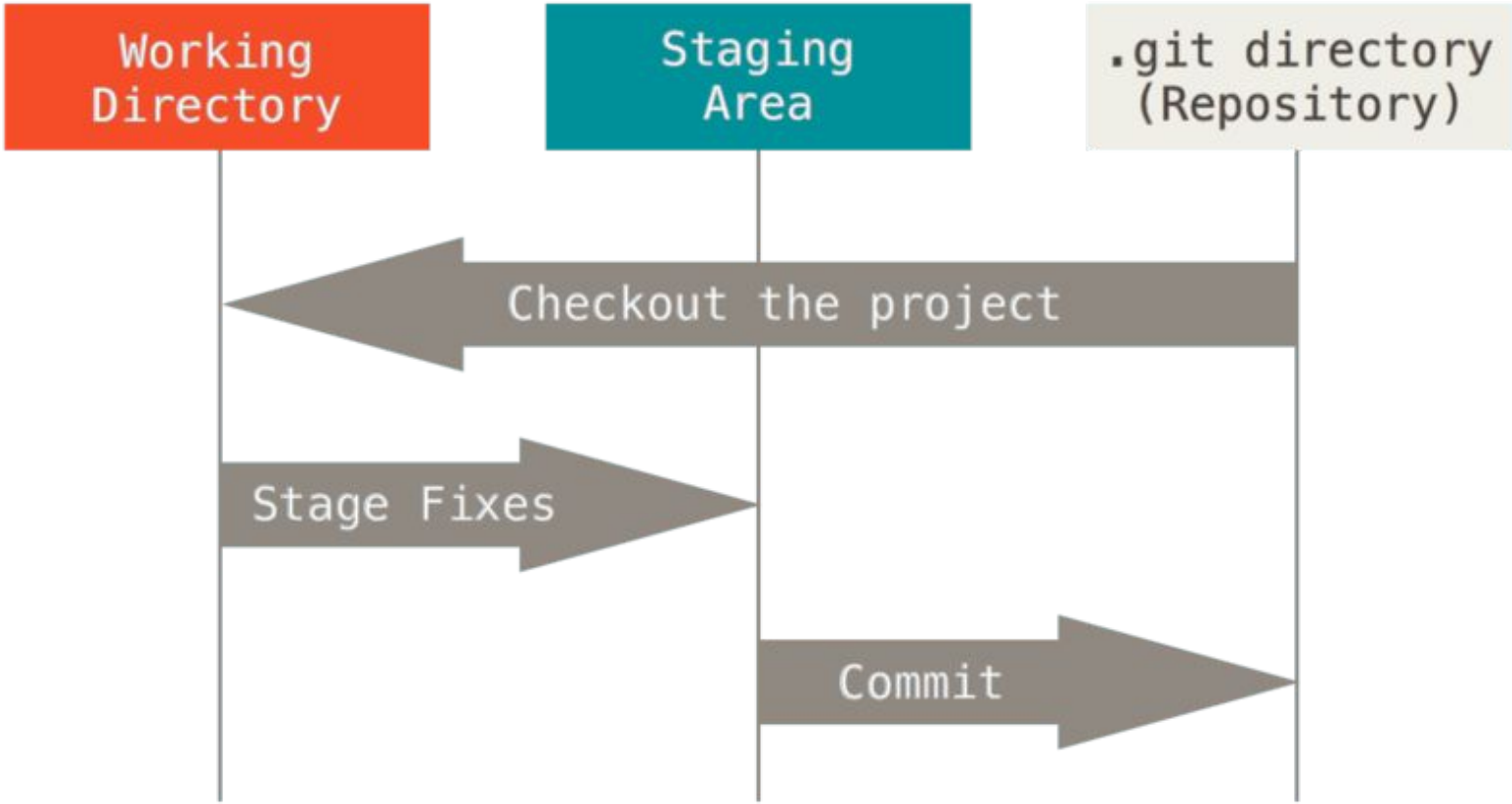
- Is a snapshot based VCS
- Was created by Linus Torvalds in 2005 for collaborative development of the Linux kernel
- He described the tool as "the stupid content tracker" and
  - "global information tracker" when it works for you
  - "goddamn idiotic truckload of sh**" when it breaks

How a Git repository looks like?

Git repository has **3 areas - places** where your text files (or info about them - metadata) are stored:

- **Working directory** - (aka **working tree**) regular directory where you edit your files
- **Staging area** - (aka **index**) metadata file where you mark which files were changed - it is used when you take a picture-snapshot
- **Git repository** itself - **database** which stores the snapshots

How a Git repository looks like?



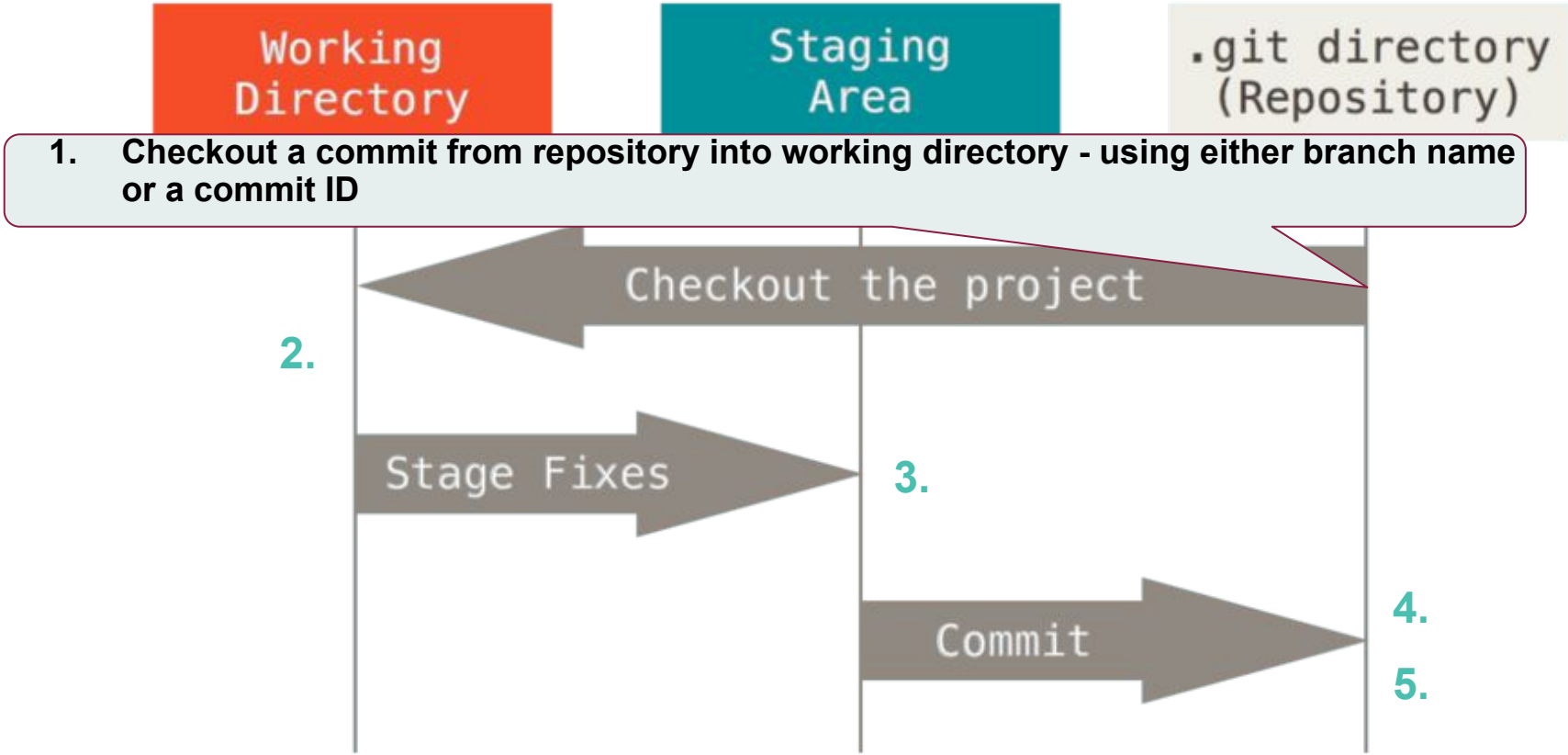
Git jargon

- **Checkout** files - extract files into working directory from existing snapshot in the repository (database)
- **Add** a file - mark a file as modified in the staging area (index)
- **Commit** (**verb**) - add (check in) all the changes marked in the staging area into the repository (database) by creating a new snapshot
- **Commit** (**noun**) - a picture-snapshot created by committing, identified by unique immutable commit ID

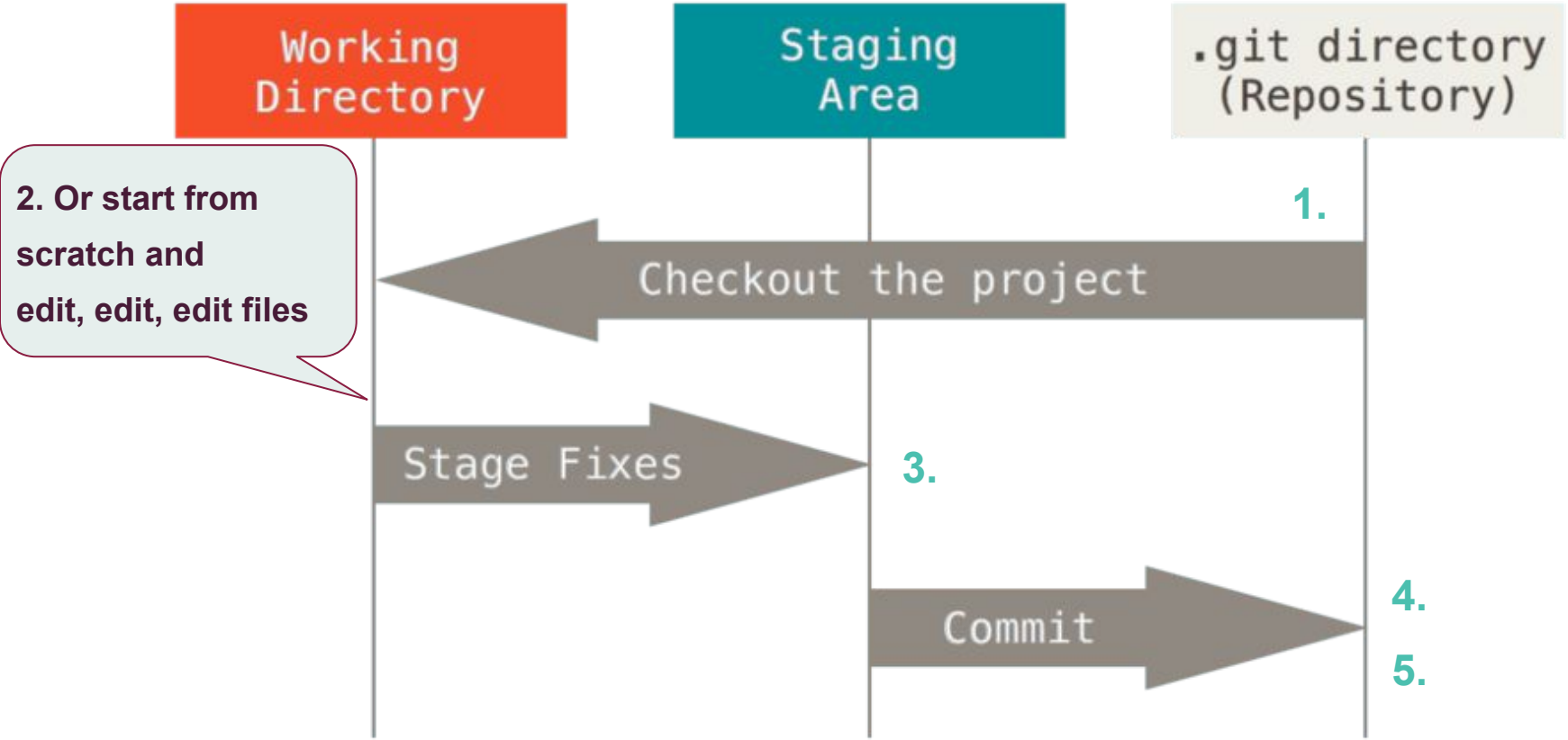
Git jargon

- **Branch** (**noun**) - human friendly name - mutable pointer to the latest snapshot - identified by commit ID

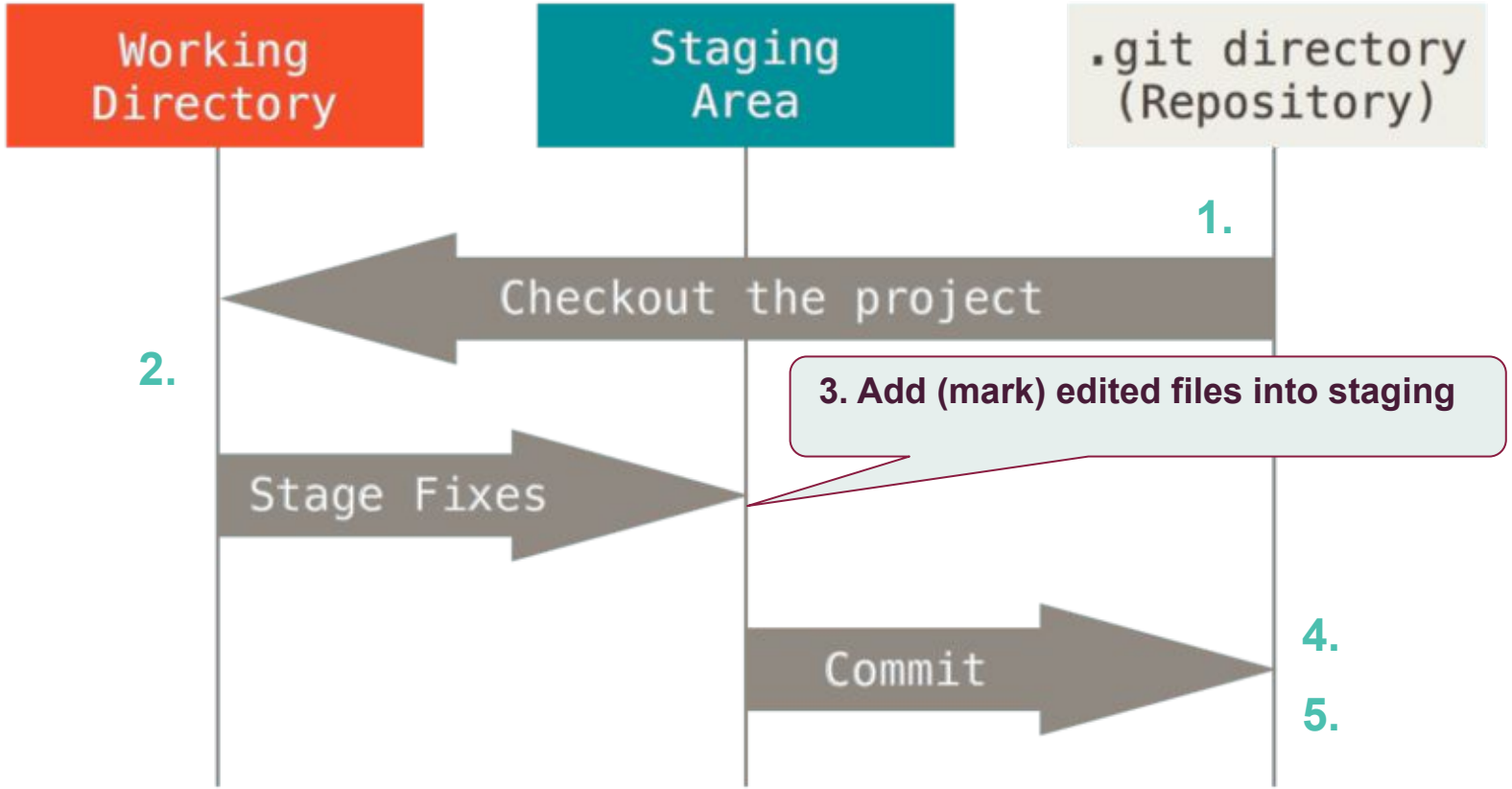
(1) Git simple workflow - checkout



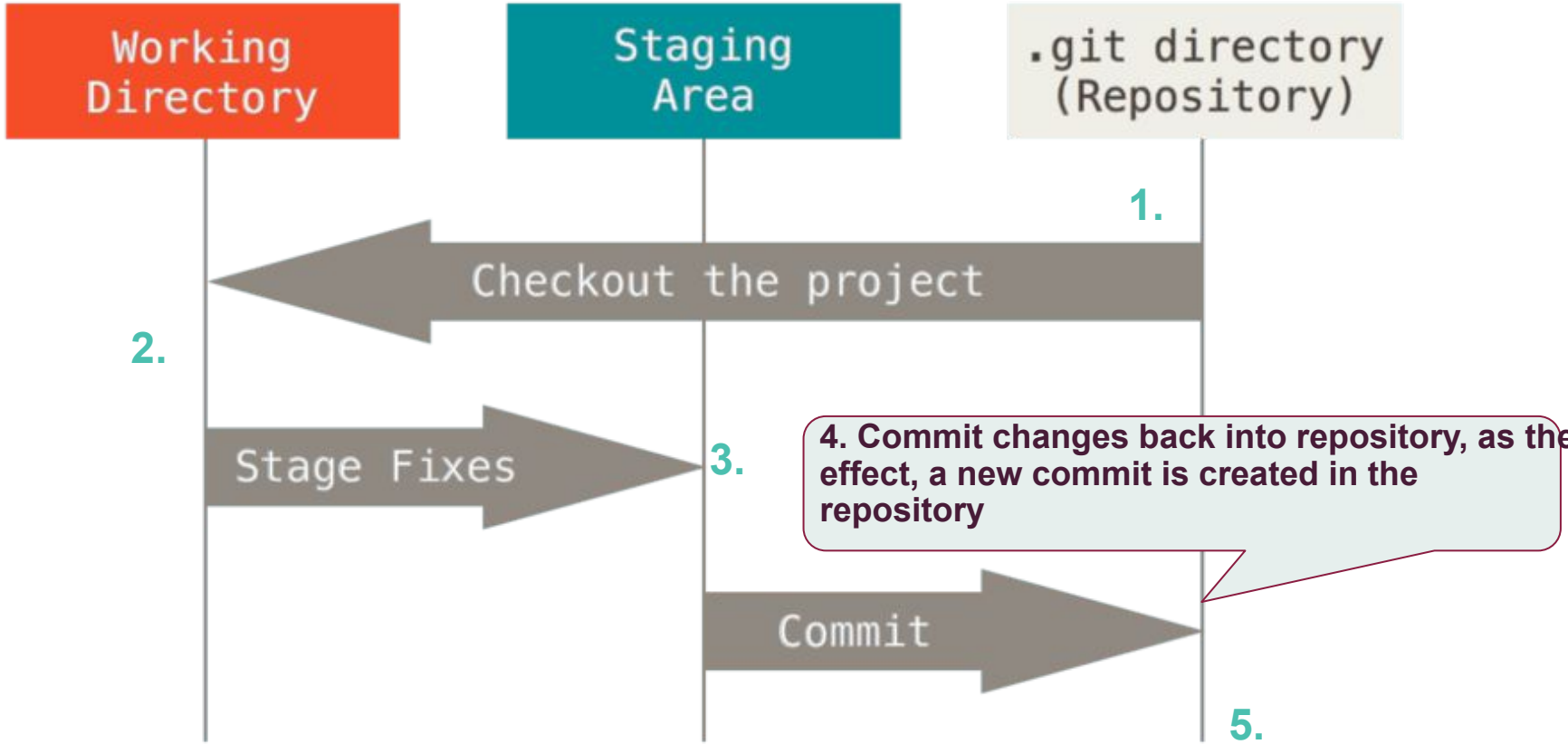
(2) Edit, edit... edit



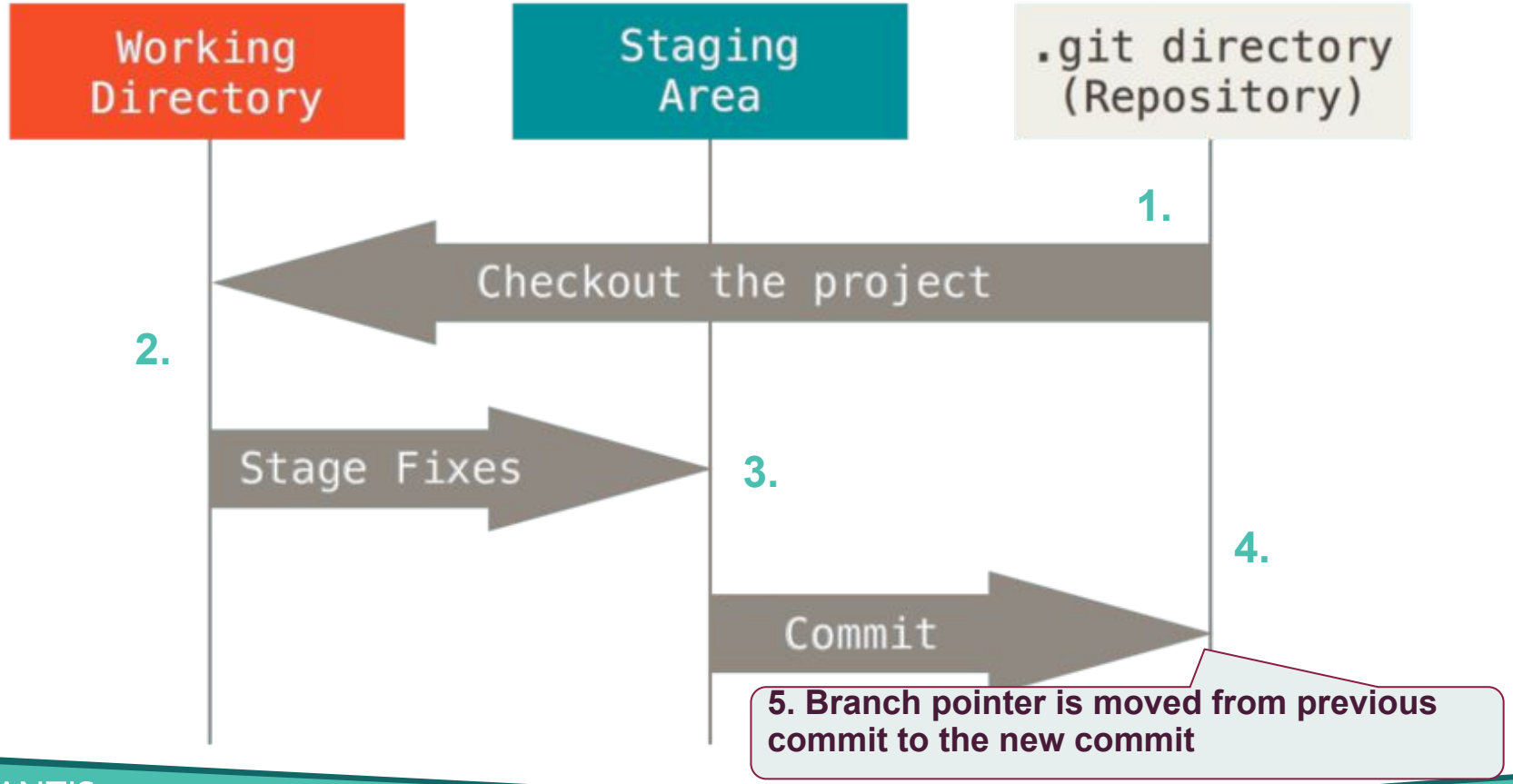
(3) Add to staging



(4) Commit


Working Directory
Staging Area
.git directory (Repository)
1.
Checkout the project
2.
Stage Fixes
3.
4. Commit changes back into repository, as the effect, a new commit is created in the repository
Commit
5.

(5) New commit is created and branch pointer is moved



Git simple workflow - summary

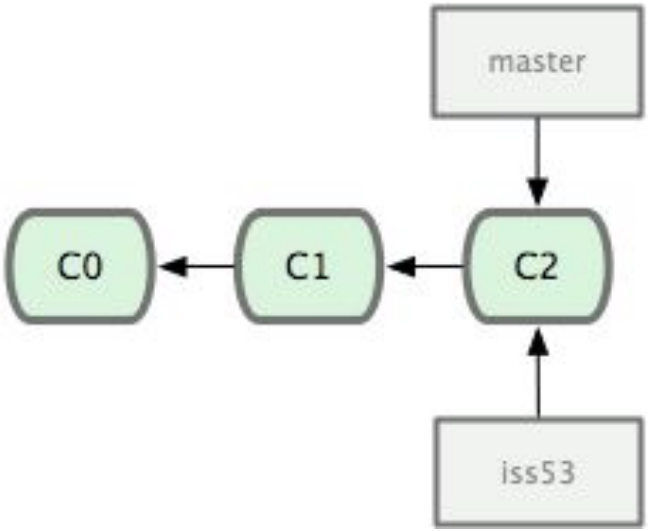
1. Checkout (extract) an existing commit (snapshot) from repository (database) into working directory - using either branch name or a commit ID or
2. Start from scratch and edit, edit... edit the files
3. Add (mark) edited files into staging
4. Commit (add/check-in) changes back into repository (database), as the effect, a new snapshot (commit) is created in the repository and
5. Branch pointer is moved from the previous commit to the new commit

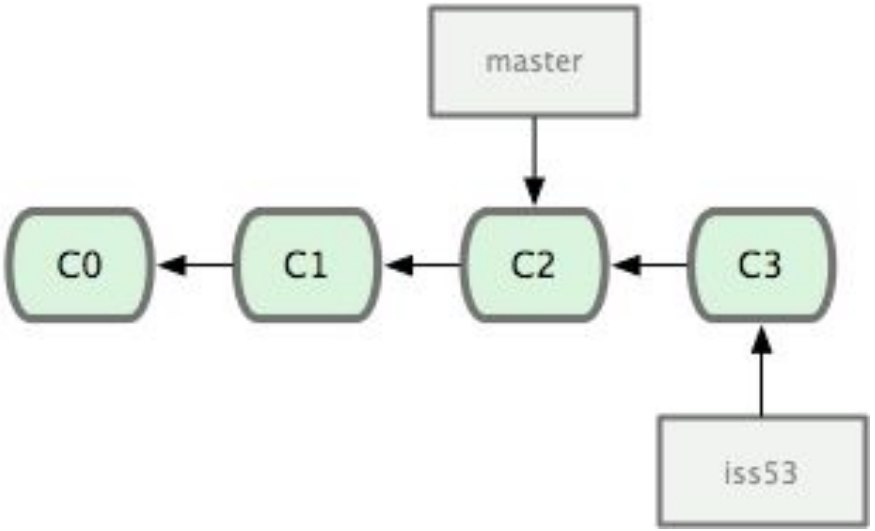
Git jargon

- **Branch** (**noun**) - human friendly name - mutable pointer to the latest snapshot - identified by commit ID
- **Branch (verb)** - create a new branch (new pointer)
- **Merge** (**verb**) - merge changes from one branch into another branch

Branch-merge example1

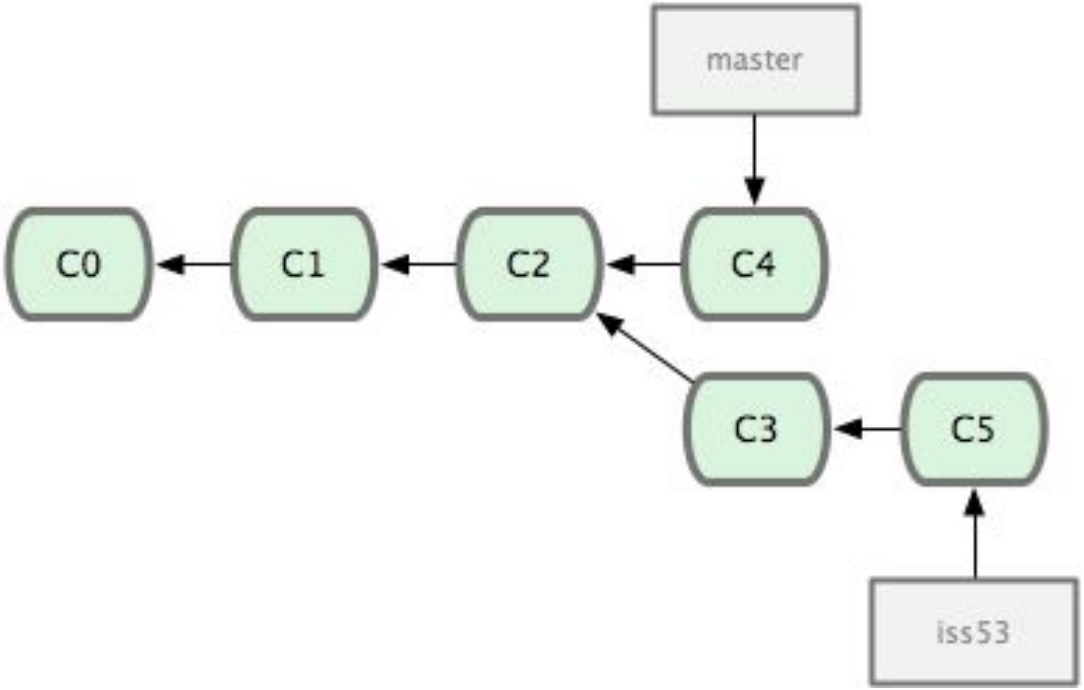
How to merge iss53 into master branch?





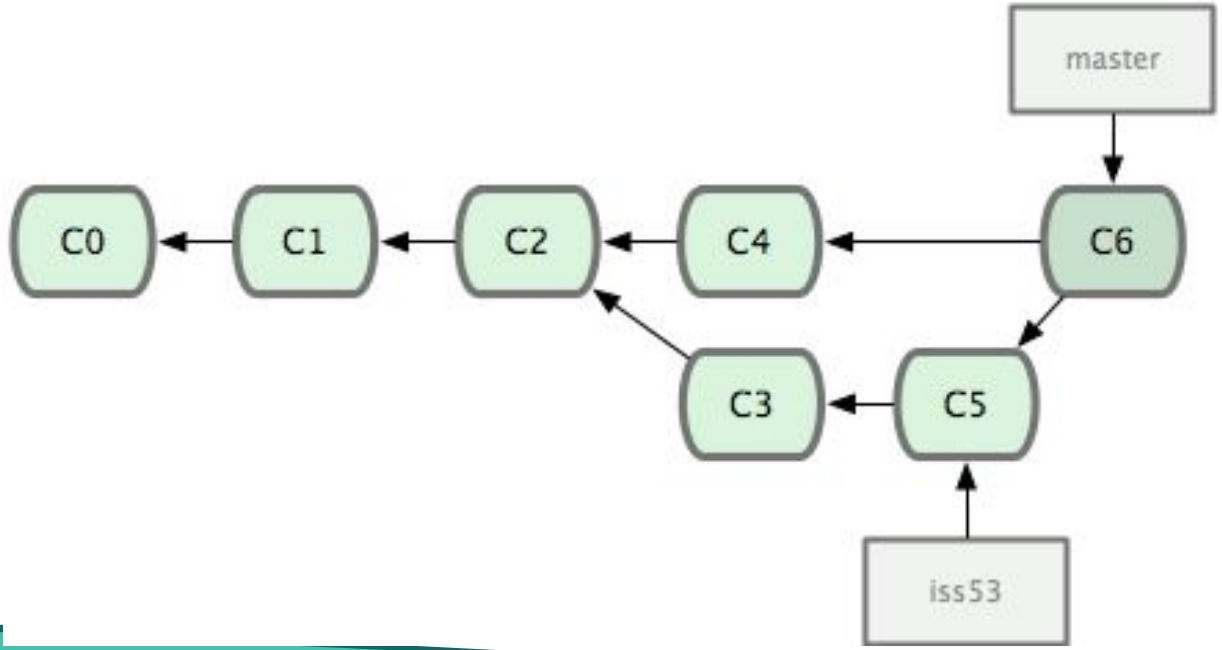
Branch-merge - example2

How to merge iss53 into master branch?



Branch-merge - example2

Git finds the best common ancestor and does the job automatically - aka 3-way marge



Collaboration with Git

- Now you have your changes stored in your repository
- But how to make your data available to other people or systems?

Collaboration with Git

- You can merge branches between repositories!
- **Push** - merge branch from your repo into branch on another repo or
- **Pull** - merge branch from another repo into branch on your repo

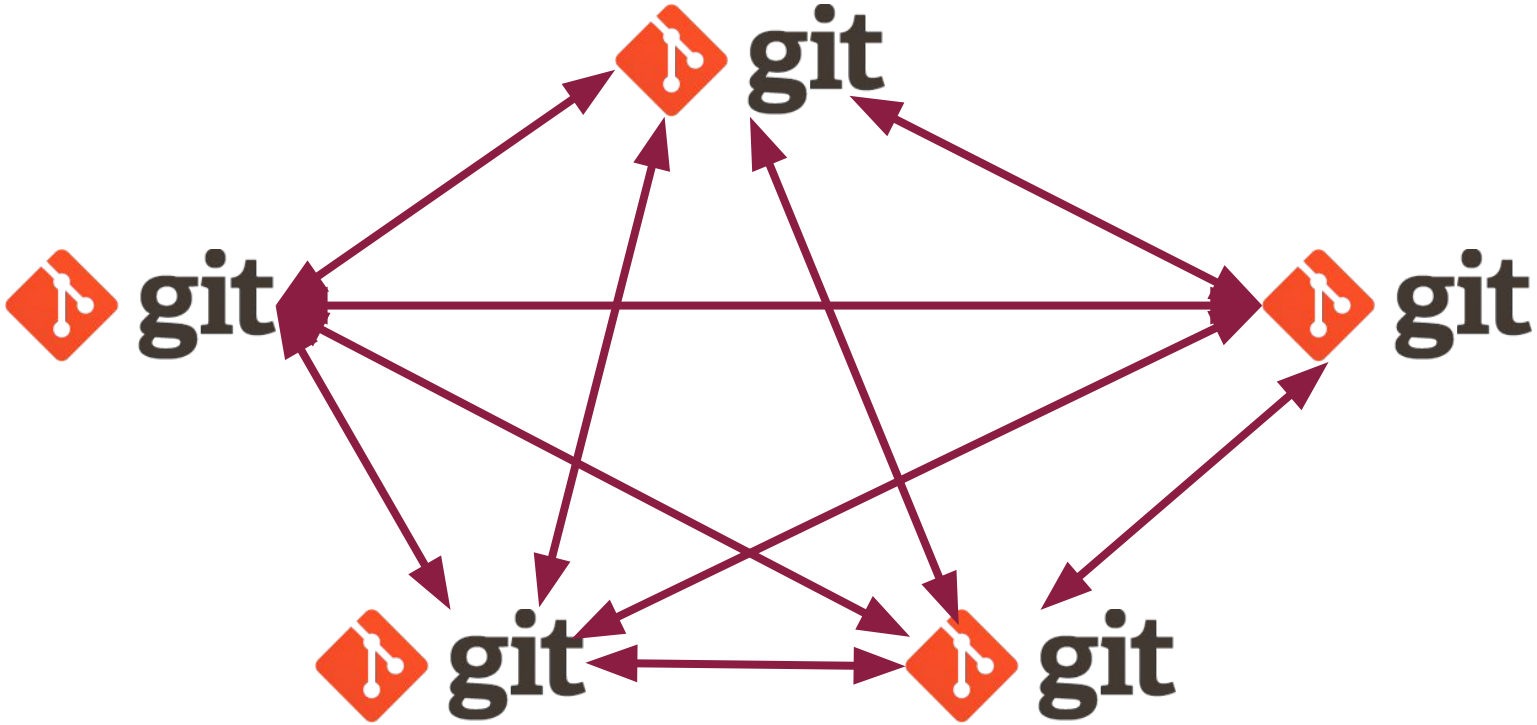
Collaboration with Git

- It’s simple
  - when you and other people have accounts on the same computer
  - or have the same file system mounted (e.g. NFS)
- Just give them:
  - **Read access** to your git repo directory - to let them pull
  - **Write access** - to let them push

Collaboration with Git

- It also works remotely over SSH protocol!
- Of course you still need the account and read/write access rights on remote computers

Collaboration with Git - peer-to-peer

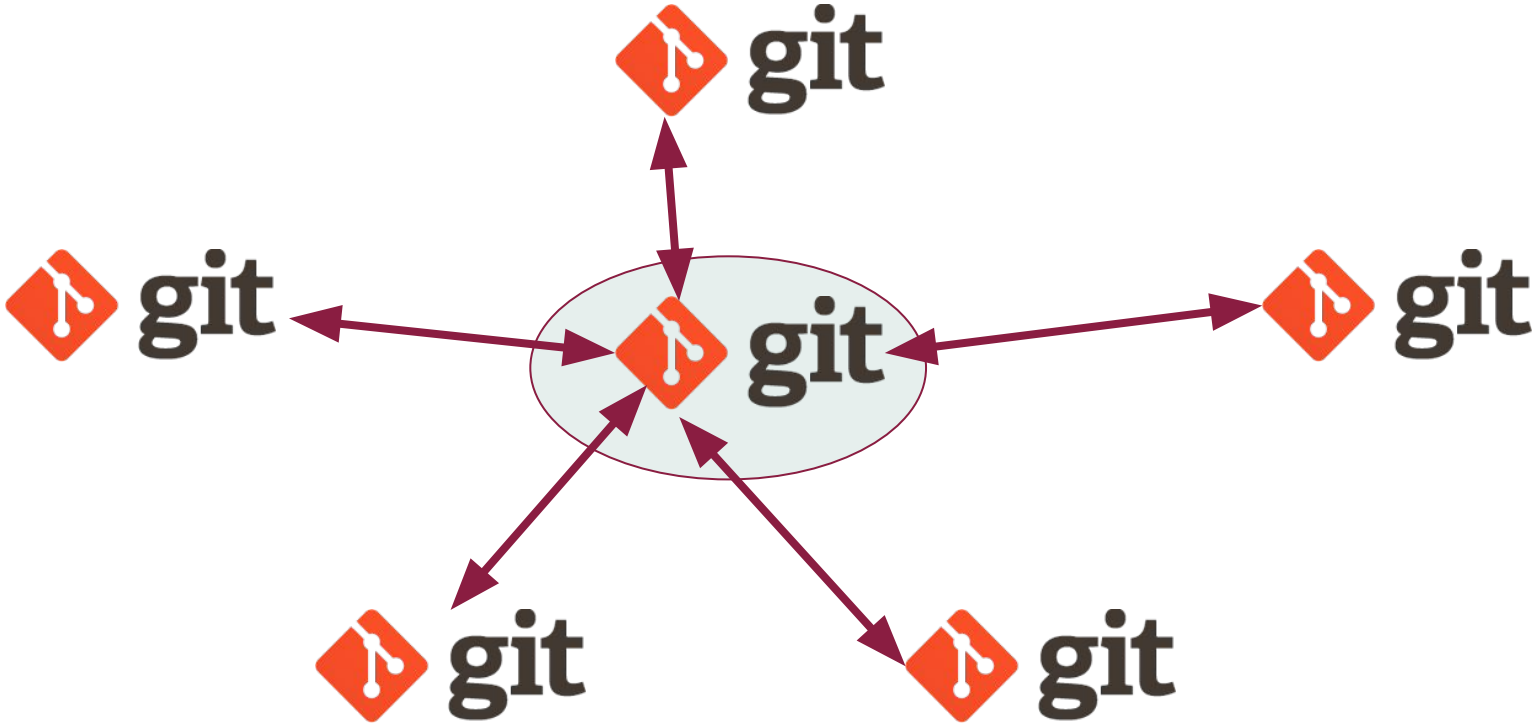


Collaboration with Git - branching

Branching in Git has two functions:

- Long-running public branches - used as a mediator for the code contributed by developers to keep high level of code stability and maturity
- Short-lived private branches - used a sandbox for the development to separate new code from mature code

Collaboration with Git - stable branch on the central repo



Collaboration with Git - is it a good idea to let them push?

- Typically you don't want everybody to push their code into your stable long-running branch without your review and testing
- But they can push when they have write permission!

Collaboration with Git - solution

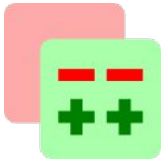
- One possible solution is not to give write access to the repository which is used to store and share stable branch and
- Appoint one developer to be a “dictator”
- Other developers send updates as patches or pull requests (e.g on mailing list)
- Once the code is accepted by the group or community (e.g by giving +1 or -1)
- The “dictator” merges the code into the stable branch

Collaboration with Git - solution

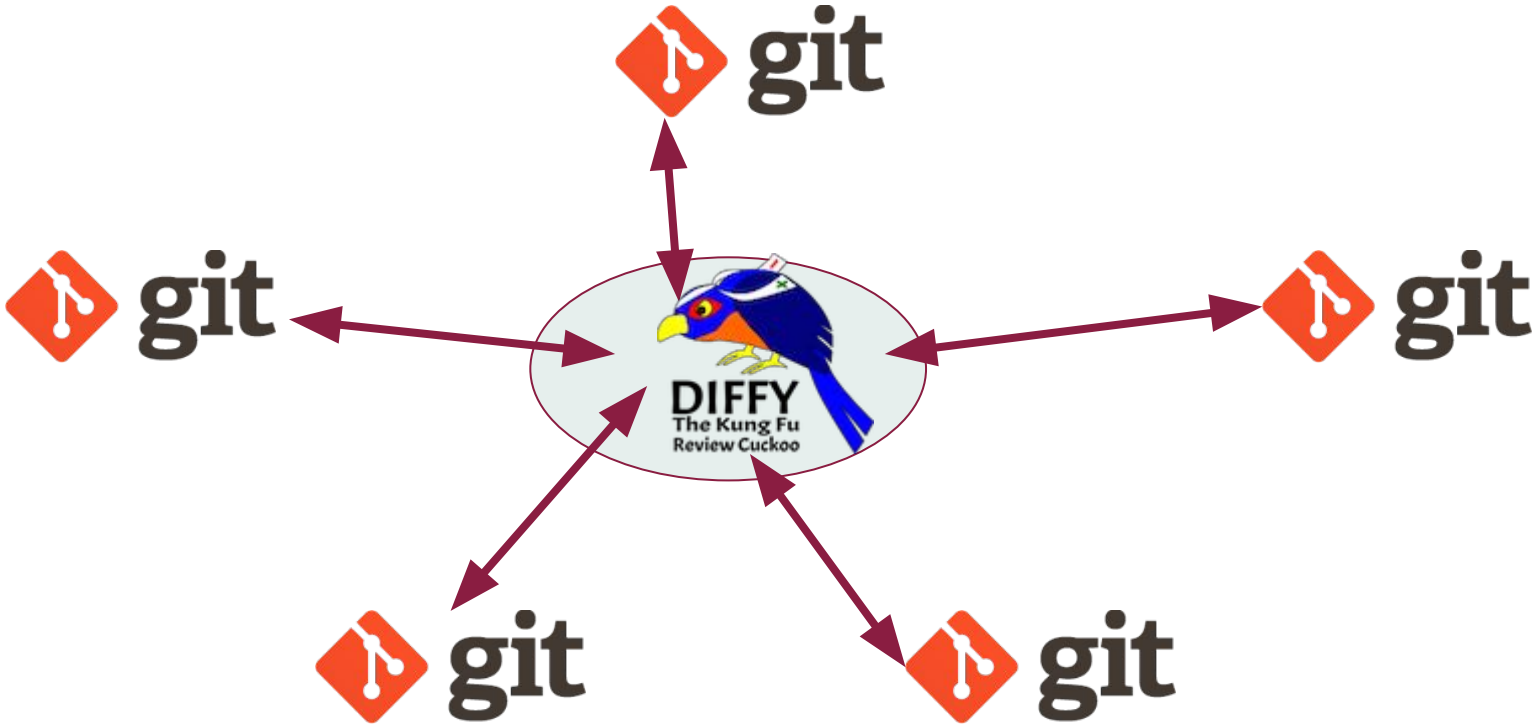
- But we don't like dictatorship
- Well, we don't like anarchy either
- We want democracy and transparency!
- The solution is...

Gerrit

Code Review for Git



Collaboration with Gerrit - stable branch on the central repo provided or gated by Gerrit



Gerrit - Code Review for Git

- Gerrit acts as a central git-ssh server with
- Web interface for reviewing-accepting commits
- It gives developers “virtual” write rights
- So they can push code
- But the code is not merged until it is reviewed and approved and (optionally) verified

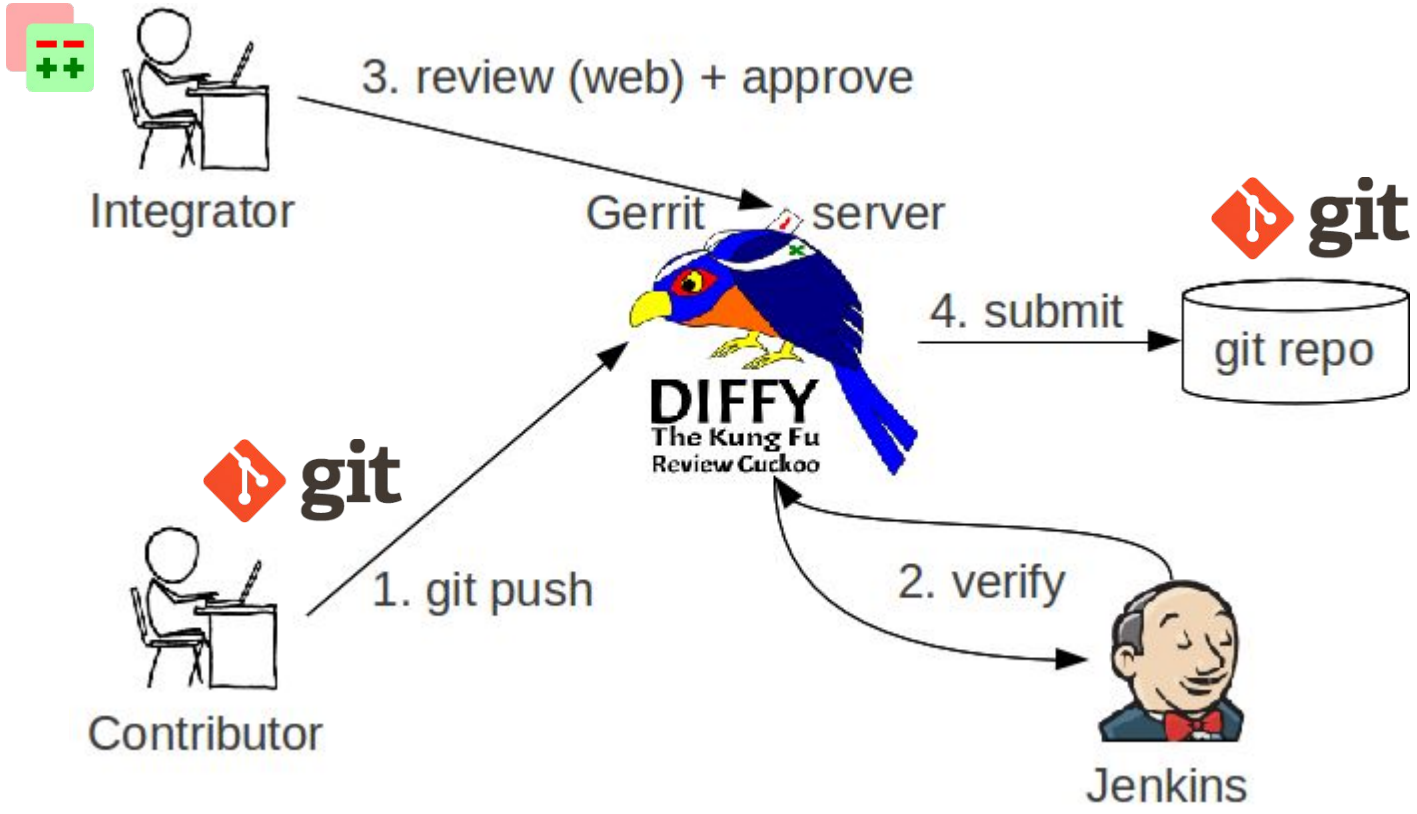
Gerrit - Code Review for Git

- Developers approve the code by voting (e.g. +1, -1)
- Verification can include e.g. syntax check and testing, and it is typically performed by automation server (not humans)

Gerrit jargon

- **Change** - identified by Change ID, allows Gerrit to track iterations of the same logical change across multiple “patchsets”
- **Patchset** is a Git commit pushed to Gerrit from developer’s repo
- **Submit** - once change is reviewed and approved (and optionally verified), developer can submit - merge their change into requested branch (e.g. using Web interface)

Collaboration with Gerrit - workflow



Jenkins

Automation server

Jenkins, at your service

Jenkins is an automation server which can be used to:

- Define jobs e.g. to
  - verify syntax of code or config
  - build, test, deliver/publish, deploy software
  - test and deploy config
- Schedule/trigger jobs execution
- Monitor jobs execution
- Present/provide results of jobs execution

Jenkins jargon

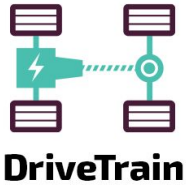
- **Project** - job definition
- **Pipeline** - our preferred type of a Jenkins project, which is defined as a DSL/Groovy script composed of stages and steps, and stored on VCS/Git
- **Build** - execution of a job/project/pipeline
- **Trigger** - event to start a build
- **Master** - Jenkins itself
- **Worker** - server where build stages/steps are executed (e.g. Salt master)

Jobs examples

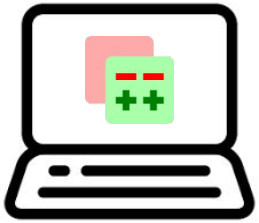
- Pipeline to verify code which was pushed to Gerrit
  - Builds are triggered by Gerrit when change is pushed
  - Result of build (success or failure) is sent back to Gerrit and used by Gerrit to accept or reject change
- Pipeline to deploy code successfully submitted-merged in Gerrit
  - Builds are triggered by Gerrit when code is submitted-merged

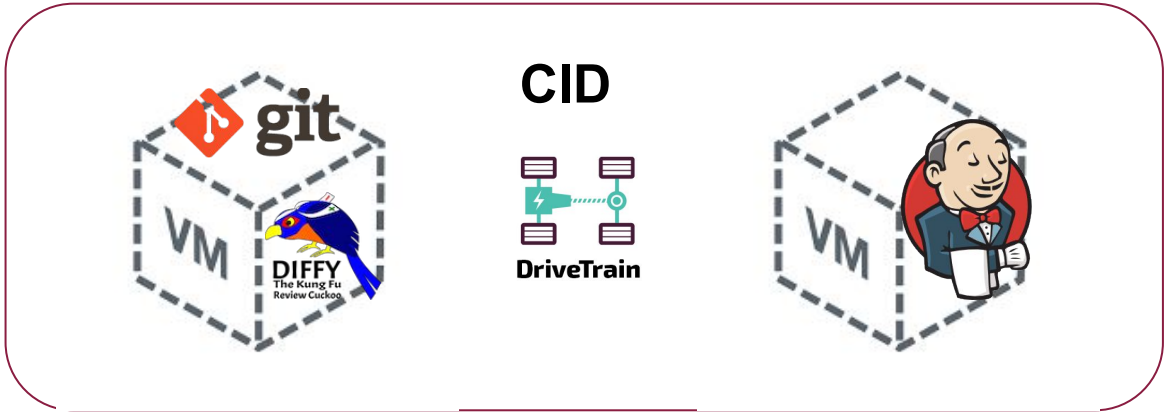
MCP DriveTrain

Git, Gerrit, Jenkins

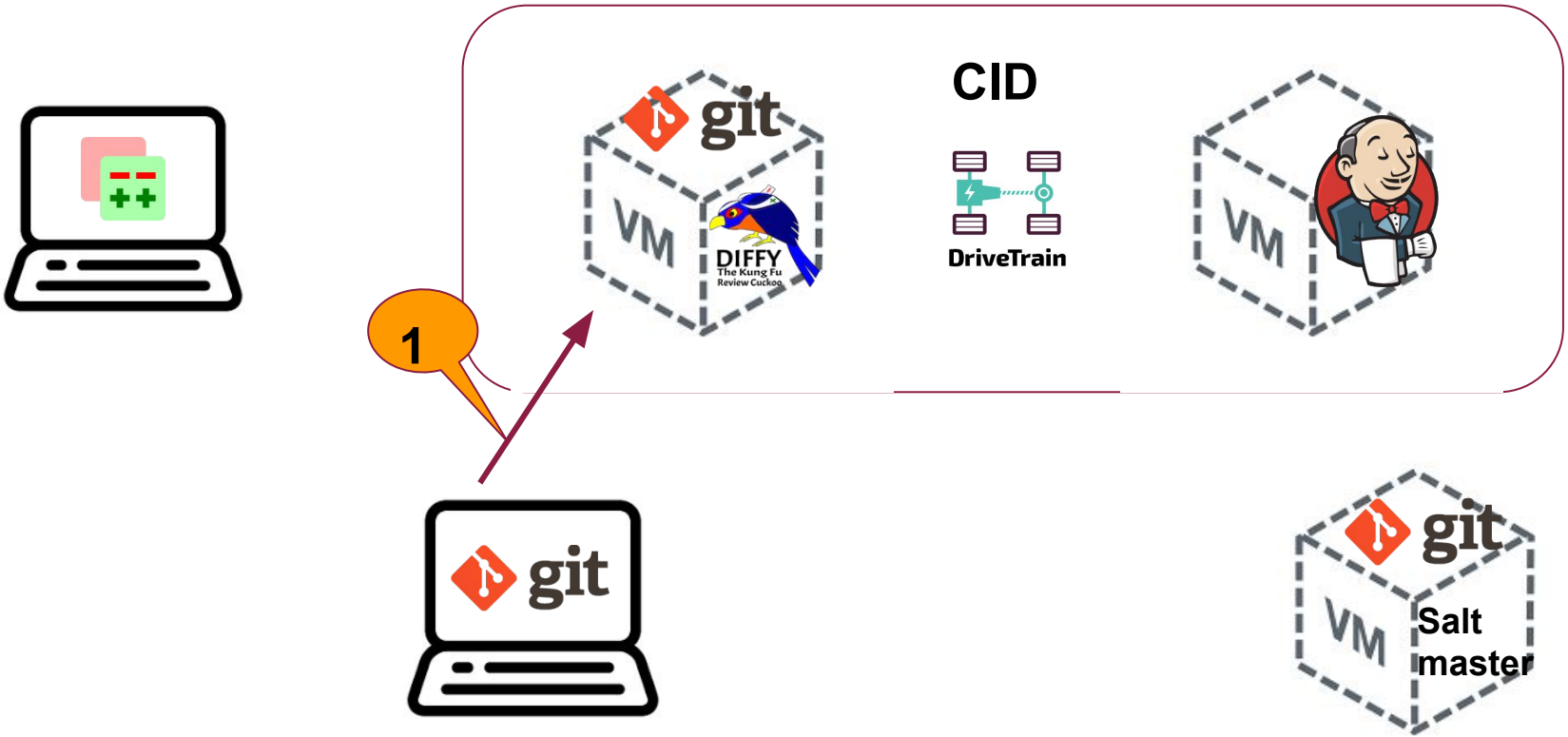


CID cluster with containers managed by Docker Swarm

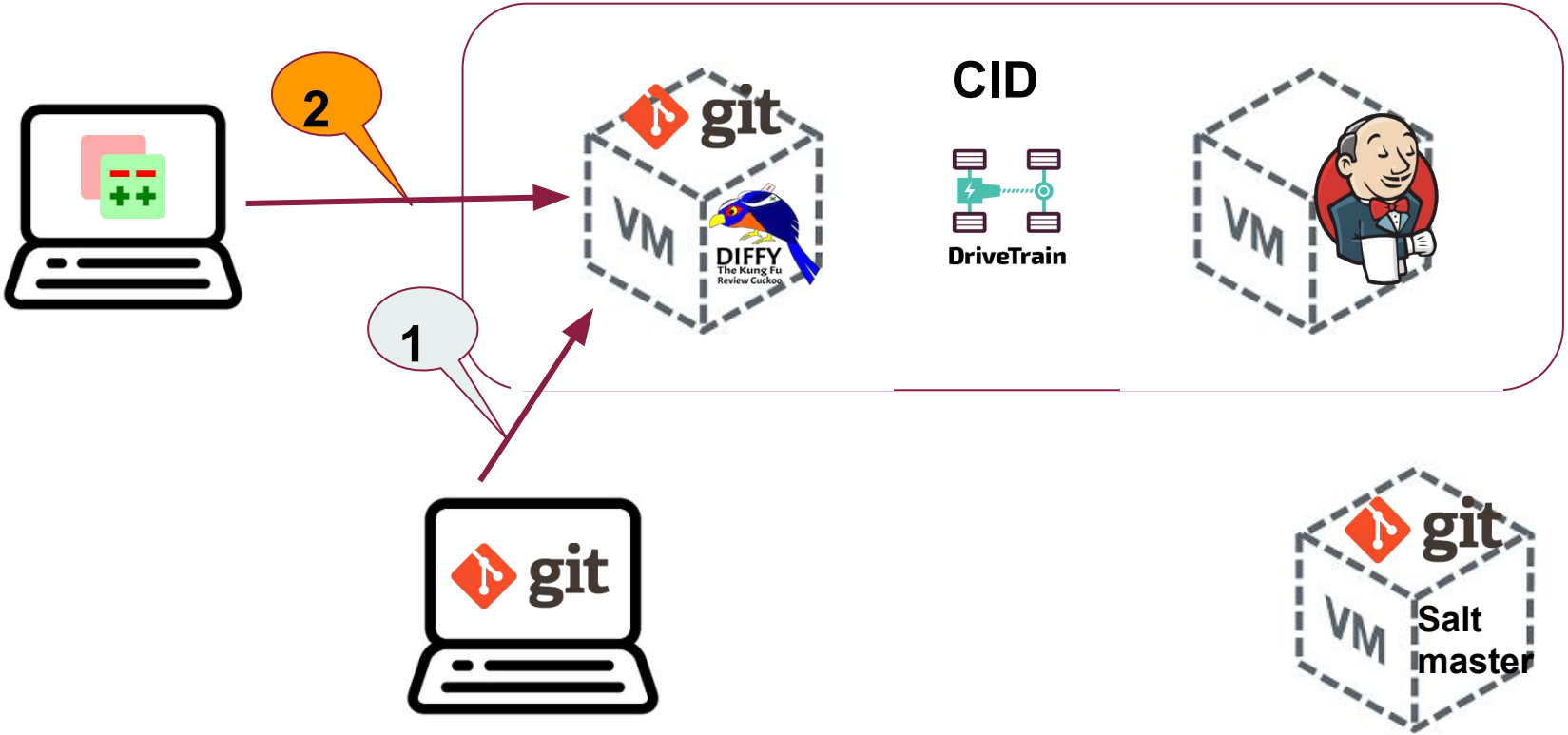




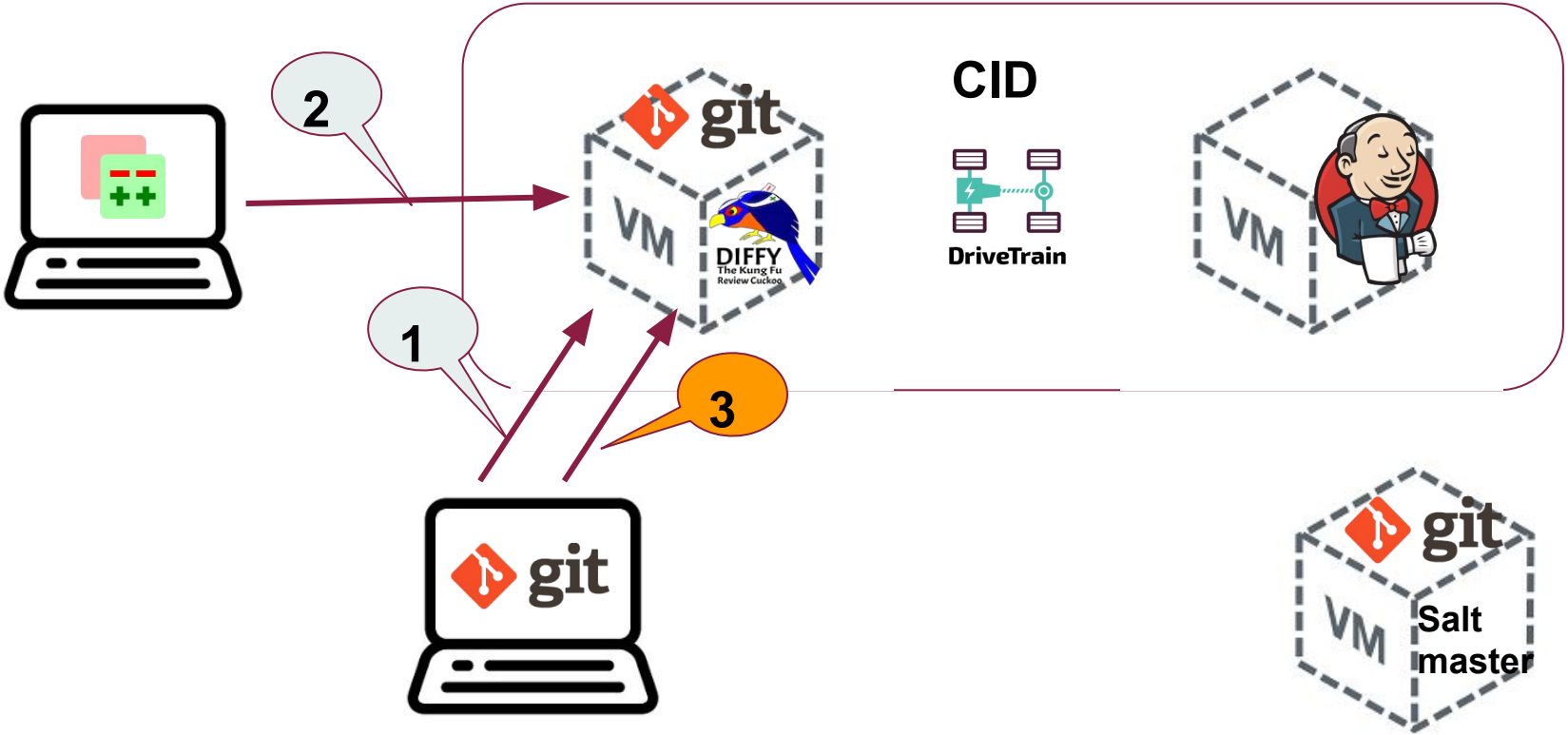
(1) Developer pushes the change from local repo to Gerrit



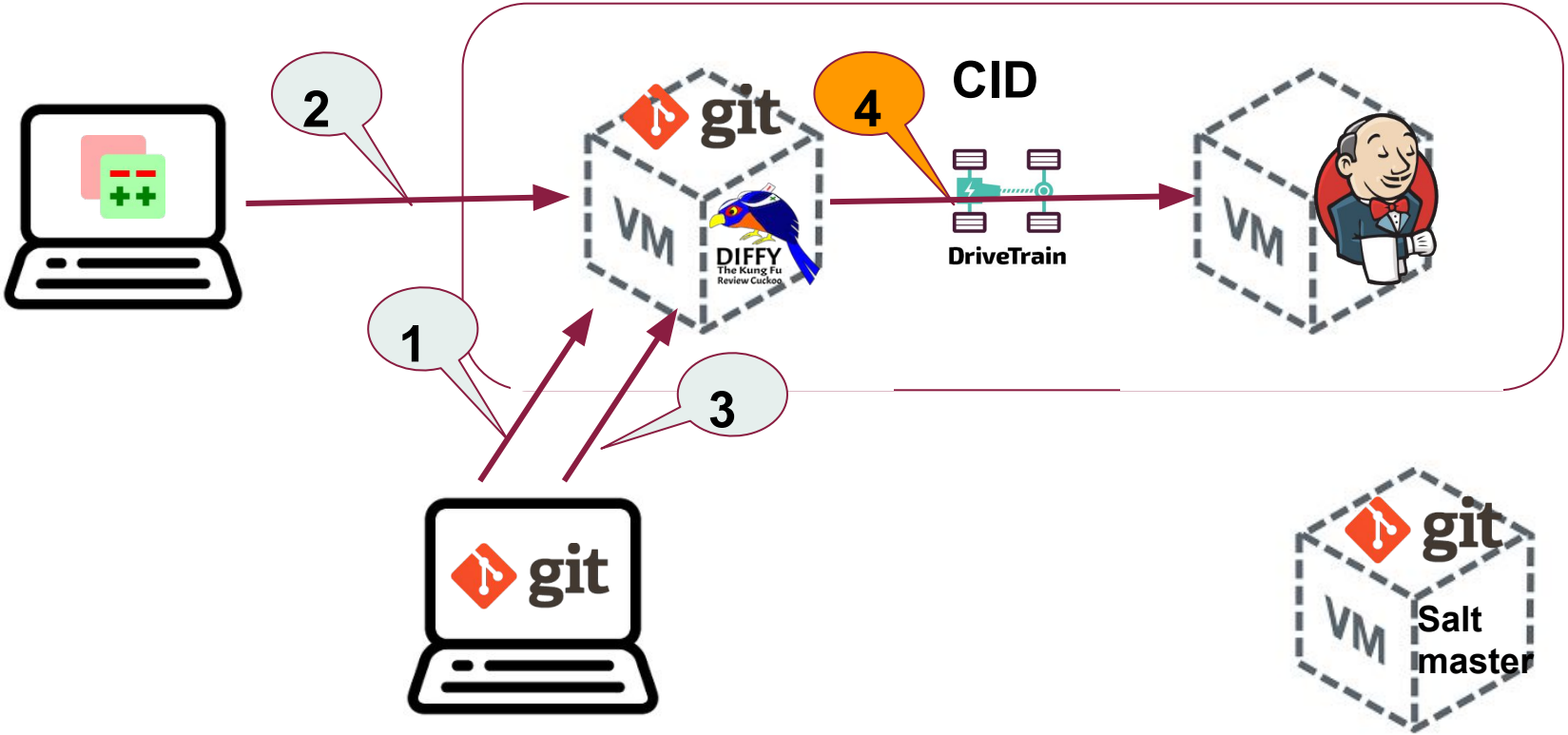
(2) Peer developers review and approve the change



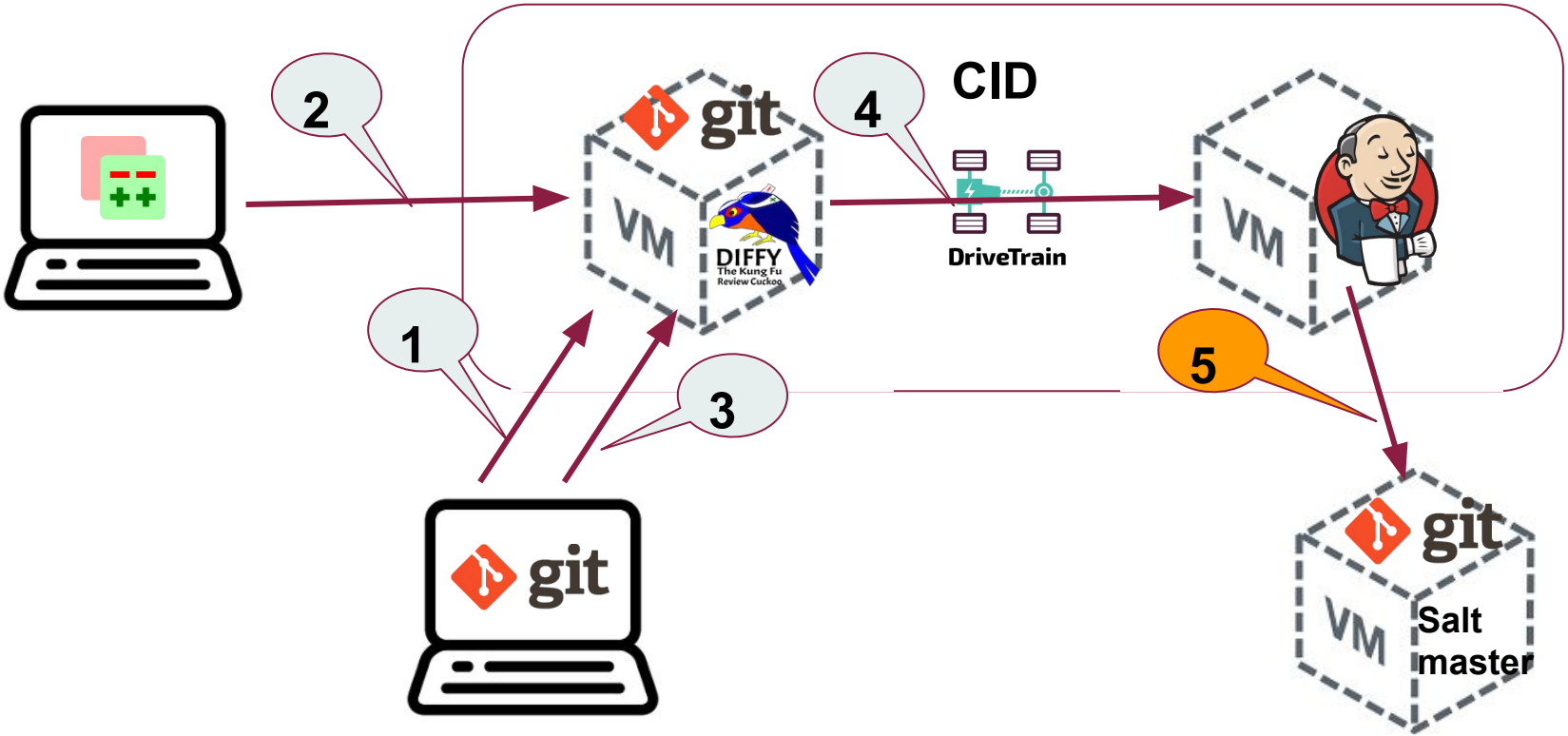
(3) Developer submit the change



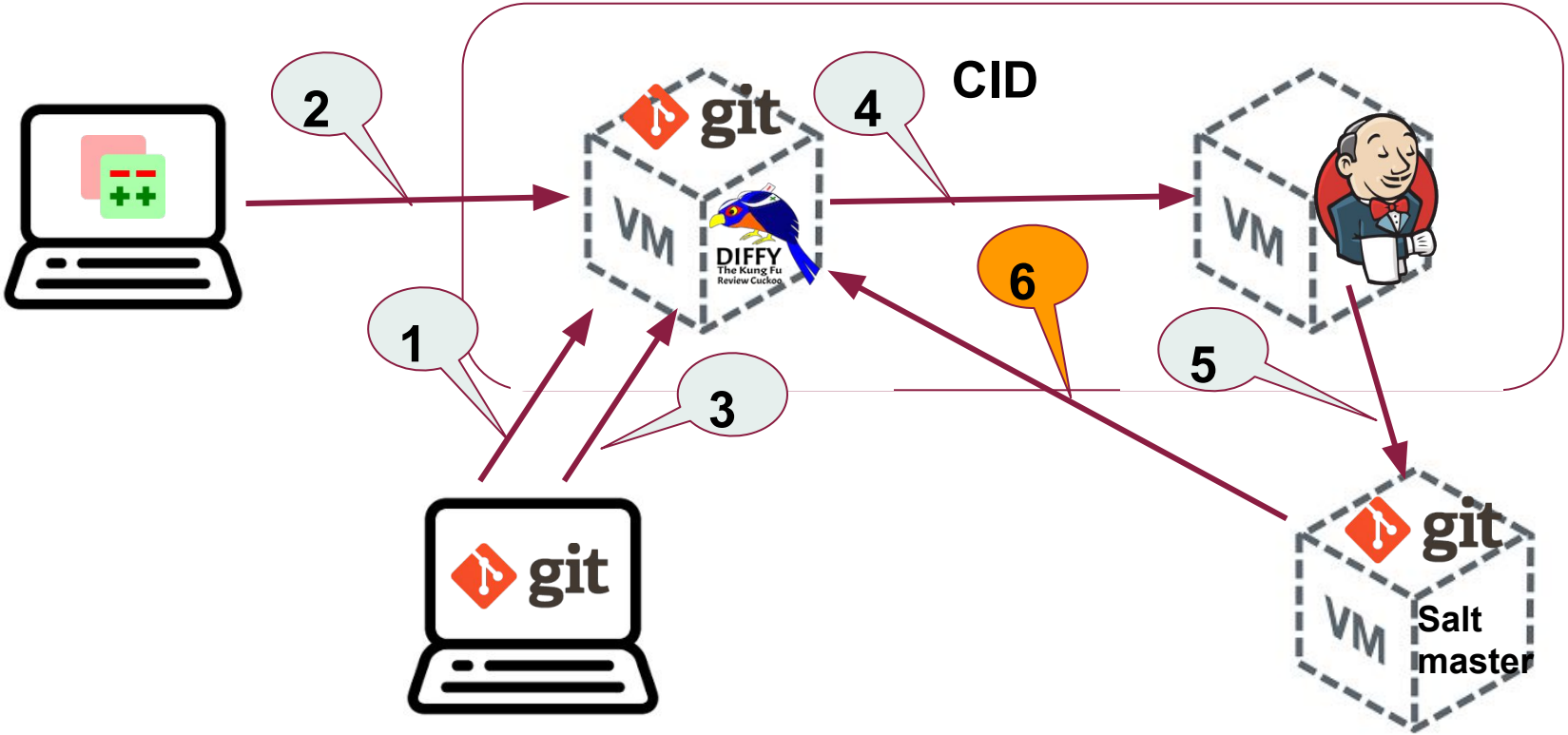
(4) Merge triggers Jenkins build (pipeline execution)



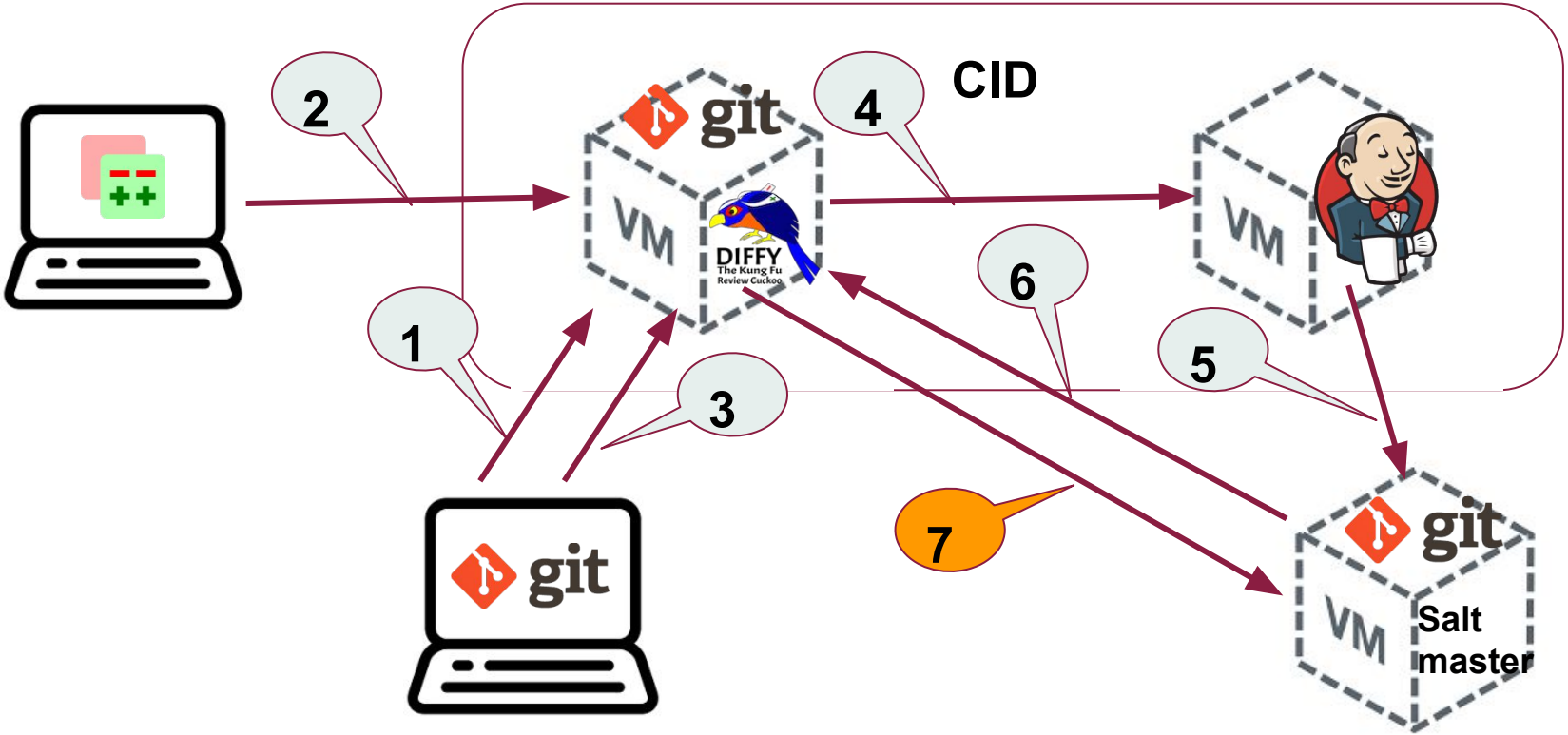
(5) Jenkins executes the build steps on Salt master



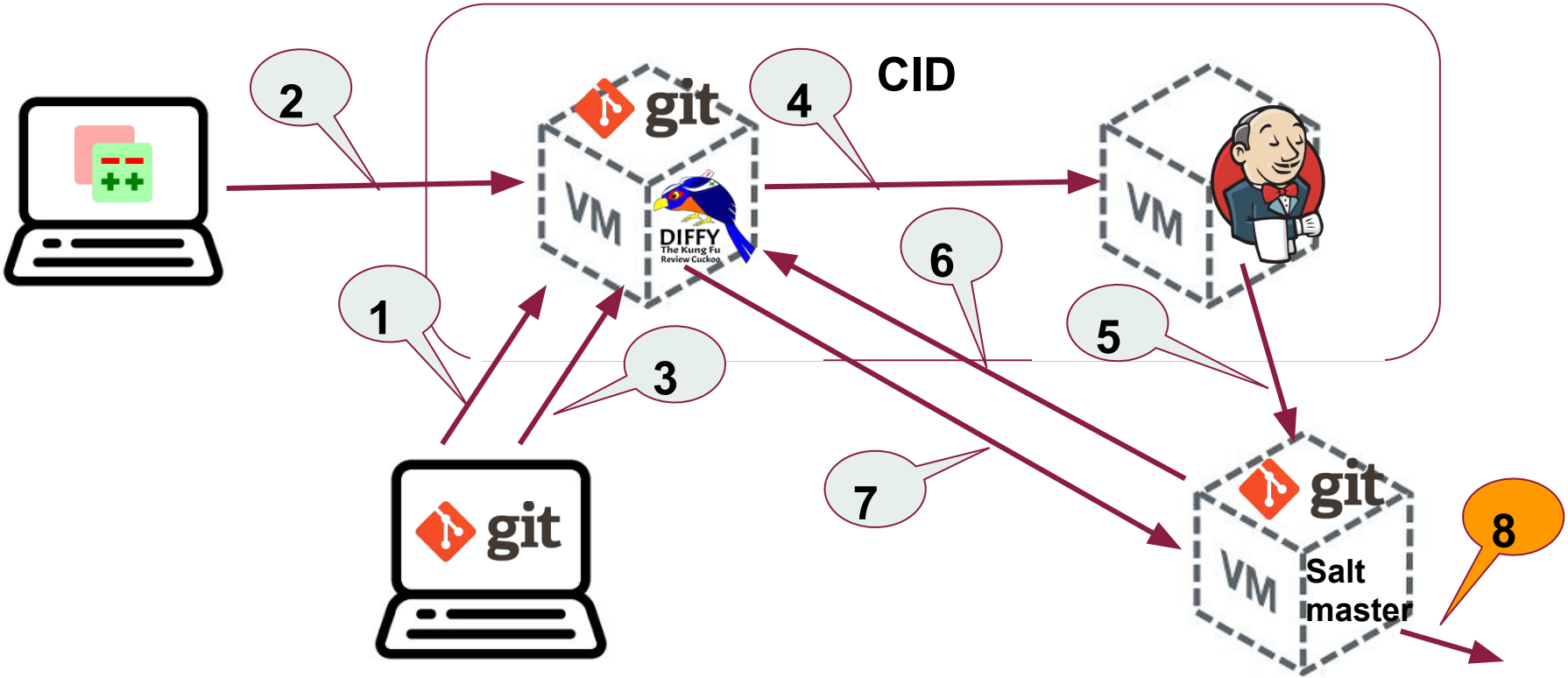
(6) Jenkins pulls the change from Gerrit to local repo on Salt master



(7) The change is pulled and merged in the Salt master repo



(8) Jenkins executes other build steps (salt calls)



Summary

- Git - let you store and share your collections of files
- Gerrit - brings democracy and transparency
- Jenkins - automates

Questions?