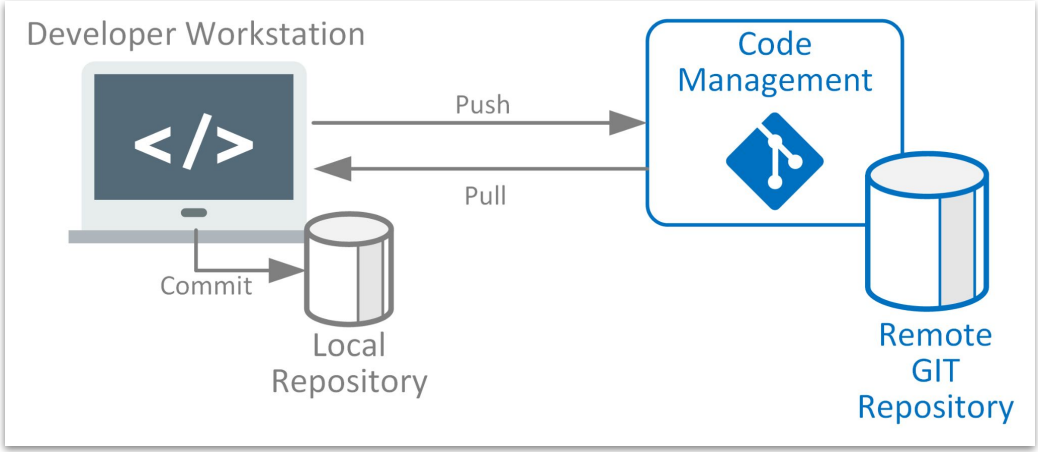
System and Network Engineering - Lecture 13

$ DevOps tools: Git and Gitlab CI/CD

Version (Source) Control

Set of tools and practices that allow to:

- ❑ Track and manage changes to the software code
- ❑ Provide changes with some kind of metadata (who, when and why)
- ❑ Keep code in a centralized storage
- ❑ Work concurrently on the same codebase
- ❑ Create a foundation for DevOps



Version Control Systems

Some software provides possibility to follow Version Control practices by:

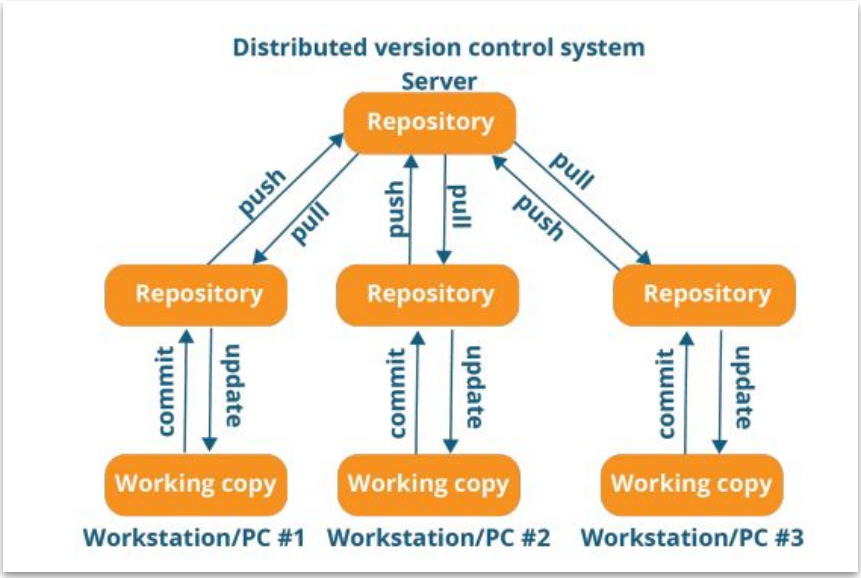
- A complete long-term change history of a file
- Branching and merging (and similar concepts)
- Having an annotated history of the code

Examples:

- GIT
- Subversion
- Perforce

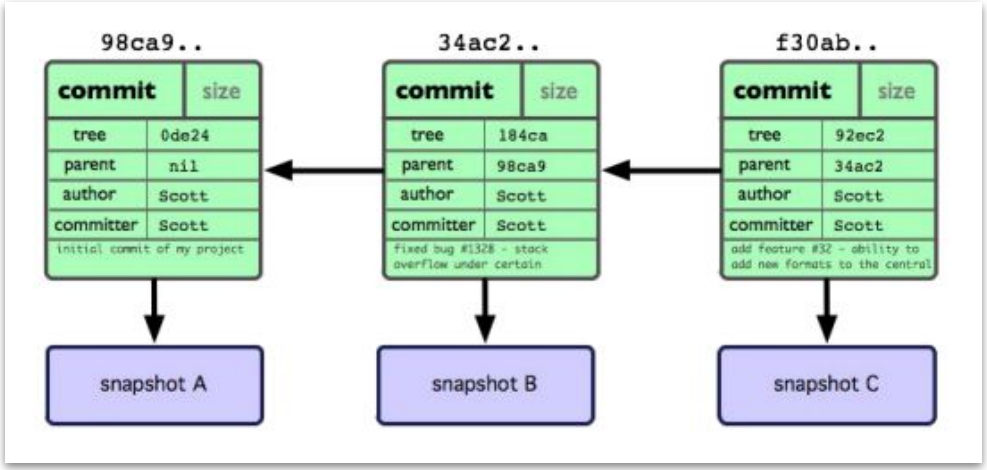
What is GIT?

- Distributed VCS
- Merge Concurrency Model (specific task - specific branch)
- Nearly every operation Is local
- Generally only adds data
- Each entity has checksum
- Lightweight
- Supported on the most of Platforms
- Free



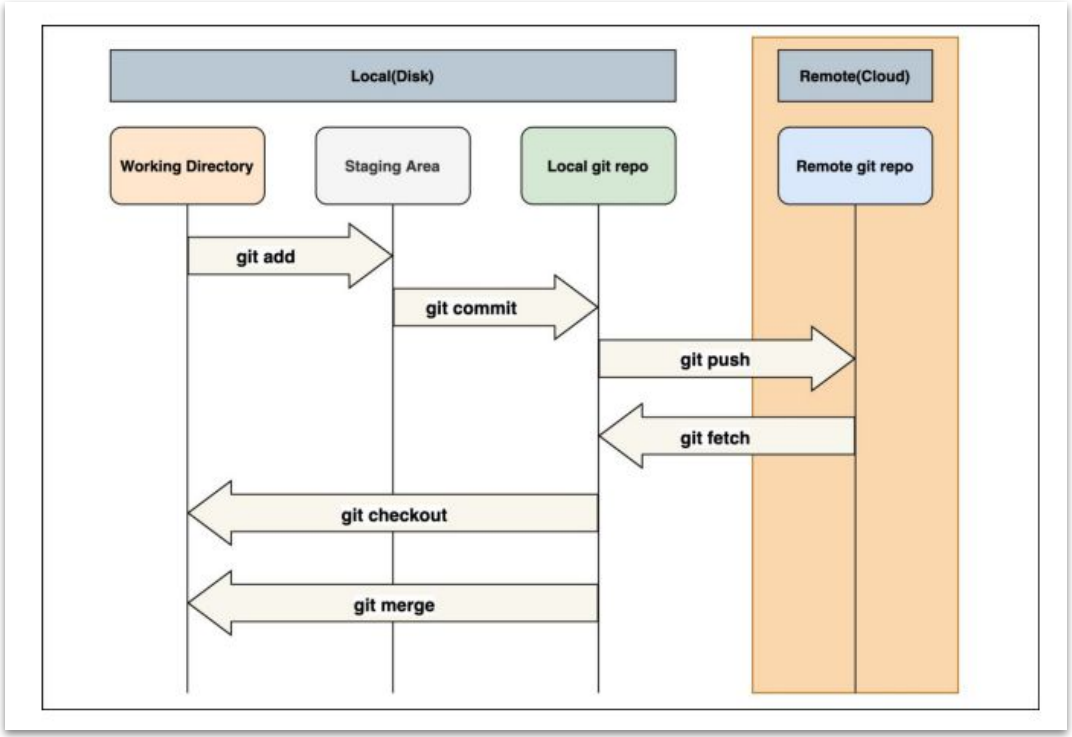
GIT Repository

- Object database
- Snapshot-based SCM
- **Commit** – recorded set of changes in the repo:
  - Current state snapshot
  - Metadata
  - Link to previous commit



Three Trees and Remote

- git add - makes files to be tracked by git from the working directory
- Once file is in the staging area:
  - you are adding its name and path to the index (special file for the tracking files)
  - git saves its current content in a object, thus starting to build the actual up-to-come commit's payload.
- Then when you do git commit, git simply saves the content of this index as a new revision, creating unique hash and metadata



Starting work with git repository

Initialize Repository:

- ❏ `$git init` - initialize directory as a git repository'
- ❏ `$git clone <remote_repo>` - create a local copy of a remote repository (i.e. download to your workspace)

Setting up your git personal information - set name and email so that the commits can associate with your user account (this config will be a global and stored in the home dir of the user `~/.gitconfig`). However, it can be replaced by the local git config configuration once you do it in the working directory without `--global` option.

- ❏ `$git config --global user.name "<name>"`
- ❏ `$git config --global user.email "email@abc.com"`

Save the current index (staging) as a snapshot and commit to the project's history

- ❏ `$git commit -m "description of the change"` - commit the changes
- ❏ `$git commit -am "description of the change"` - automatically add all modified files to the staging and commits
- ❏ `$git commit --amend` - allows to modify last commit and log message (reverse it back)

GIT Objects

- **`git` is a content-addressable filesystem** - which stores data as a key-value pair. You can retrieve the content by the key

```
saltanov@lp-0592:~/study/gitlab_ci/gitlab-cicd-exercise/.git/objects$ cd 0c/
saltanov@lp-0592:~/study/gitlab_ci/gitlab-cicd-exercise/.git/objects/0c$ git cat-file -t 0cd6396d87983c1c1308ad00e2f827e1e9851037
blob
saltanov@lp-0592:~/study/gitlab_ci/gitlab-cicd-exercise/.git/objects/0c$ git cat-file -p 0cd6396d87983c1c1308ad00e2f827e1e9851037
#name of the job. Job must contain at least the script clause. script specify the commands to execute
run_tests:
  image: python:3.9-slim-buster
  before_script:
    - apt-get update && apt-get install make
  script:
    - make test
```

e.g. of the blob object named as SHA-1 hash and which stores the snapshot of the previously committed file (the subdirectory is named with the first 2 characters of the SHA-1)

GIT Objects

- `git` stores different types of objects in `.git/objects`. The object types are:
  - commit;
  - tree;
  - blob;
  - annotated tag.

As we can see in the example when we committed a new file, 3 objects were created:

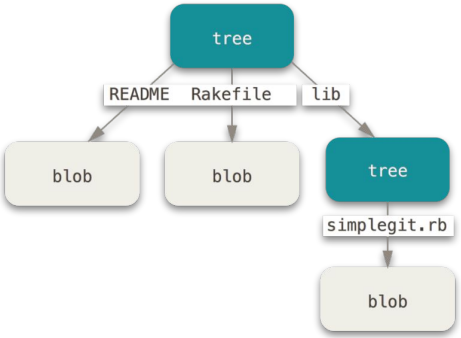
1. **Commit object type -** the commit object contains the directory tree object hash, parent commit hash, author, committer, date and message.

```
saltanov@lp-0592:~/study/test_git$ git log
commit cb8476f903326070640a4770171c1f76f9fbff27 (HEAD -> master)
Author: Saltanov Kirill <saltanovlive@gmail.com>
Date:   Sun Nov 20 23:04:05 2022 +0400

    new commit
saltanov@lp-0592:~/study/test_git$ git cat-file -p
.git/     test.txt
saltanov@lp-0592:~/study/test_git$ git cat-file -p  cb8476f903326070640a4770171c1f76f9fbff27
tree 53da65fc3517a69a470ea9bfaab47ef4087ea20d
author Saltanov Kirill <saltanovlive@gmail.com> 1668971045 +0400
committer Saltanov Kirill <saltanovlive@gmail.com> 1668971045 +0400

new commit
```

```
saltanov@lp-0592:~/study/test_git/.git$ cd objects/
saltanov@lp-0592:~/study/test_git/.git/objects$ ll
total 16
drwxr-xr-x 4 saltanov saltanov 4096 Nov 20 23:02 ./
drwxr-xr-x 7 saltanov saltanov 4096 Nov 20 23:02 ../
drwxr-xr-x 2 saltanov saltanov 4096 Nov 20 23:02 info/
drwxr-xr-x 2 saltanov saltanov 4096 Nov 20 23:02 pack/
saltanov@lp-0592:~/study/test_git/.git/objects$ echo "123" > ../../test.txt
saltanov@lp-0592:~/study/test_git/.git/objects$ git add ../../test.txt
fatal: this operation must be run in a work tree
saltanov@lp-0592:~/study/test_git/.git/objects$ cd ../../
saltanov@lp-0592:~/study/test_git$ git add test.txt
saltanov@lp-0592:~/study/test_git$ git commit -m "new commit"
[master (root-commit) cb8476f] new commit
 1 file changed, 1 insertion(+)
 create mode 100644 test.txt
saltanov@lp-0592:~/study/test_git$ ll .git/objects/
total 28
drwxr-xr-x 7 saltanov saltanov 4096 Nov 20 23:04 ./
drwxr-xr-x 8 saltanov saltanov 4096 Nov 20 23:04 ../
drwxr-xr-x 2 saltanov saltanov 4096 Nov 20 23:03 19/
drwxr-xr-x 2 saltanov saltanov 4096 Nov 20 23:04 53/
drwxr-xr-x 2 saltanov saltanov 4096 Nov 20 23:04 cb/
drwxr-xr-x 2 saltanov saltanov 4096 Nov 20 23:02 info/
drwxr-xr-x 2 saltanov saltanov 4096 Nov 20 23:02 pack/
saltanov@lp-0592:~/study/test_git$ tree .git/objects/
.git/objects/
├── 19
│   └── 0a18037c64c43e6b11489df4bf0b9eb6d2c9bf
├── 53
│   └── da65fc3517a69a470ea9bfaab47ef4087ea20d
├── cb
│   └── 8476f903326070640a4770171c1f76f9fbff27
├── info
└── pack

5 directories, 3 files
```

GIT Objects

2. **Tree object type -** tree object contains one line per file or subdirectory, with each line giving file permissions, object type, object hash and filename. Object type is usually one of "blob" for a file or "tree" for a subdirectory



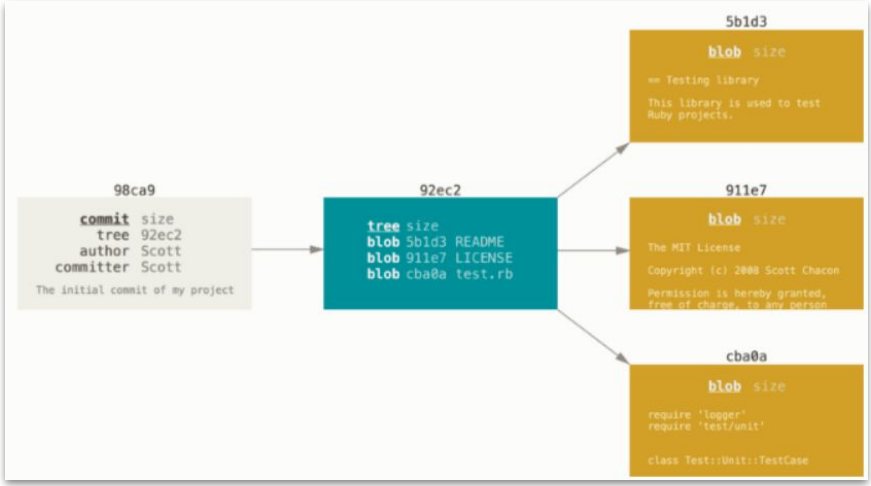

```
$ git cat-file -p master^{tree}
100644 blob a906cb2a4a904a152e80877d4088654daad0c859      README
100644 blob 8f94139338f9404f26296befa88755fc2598c289      Rakefile
040000 tree 99f1a6d12cb4b6f19c8655fca46c3ecf317074e0      lib
```

```
saltanov@lp-0592:~/study/test_git$ git cat-file -p 53da65fc3517a69a470ea9bfaab47ef4087ea20d
100644 blob 190a18037c64c43e6b11489df4bf0b9eb6d2c9bf    test.txt
saltanov@lp-0592:~/study/test_git$ git cat-file -t 53da65fc3517a69a470ea9bfaab47ef4087ea20d
tree
```

From the previous data, we received from the commit contents - it gave us the hash of the directory listing for the commit. If we inspect this object, we find it is of type "tree" and contains the directory listing for the commit:

GIT Objects

3. **Blob object type -** Blob is an abbreviation for "binary large object". When we `$git add <a file>` such as test.txt, git creates a blob object containing the contents of the file. Blobs are therefore the git object type for storing files.




```
saltanov@lp-0592:~/study/test_git$ git cat-file -t 190a18037c64c43e6b11489df4bf0b9eb6d2c9bf
blob
saltanov@lp-0592:~/study/test_git$ git cat-file -p 190a18037c64c43e6b11489df4bf0b9eb6d2c9bf
123
```

The directory listing gave us the hash of the stored of test.txt. This object is of type "blob" and contains the file snapshot

GIT Objects

4. **Tag object type (optional) -** The tag object type contains the hash of the tagged object, the type of tagged object (usually a commit), the tag name, author, date and message:

```
saltanov@lp-0592:~/study/test_git$ git tag -a v1.0 -m "my version 1.0"
saltanov@lp-0592:~/study/test_git$ tree .git/objects/
.git/objects/
├── 19
│   └── 0a18037c64c43e6b11489df4bf0b9eb6d2c9bf
├── 53
│   └── da65fc3517a69a470ea9bfaab47ef4087ea20d
├── c4
│   └── 64e7bb7d2ef08d74957199c2e367abd1bbb8cc
├── cb
│   └── 8476f903326070640a4770171c1f76f9fbff27
├── info
└── pack

6 directories, 4 files
saltanov@lp-0592:~/study/test_git$ git cat-file -t c464e7bb7d2ef08d74957199c2e367abd1bbb8cc
tag
saltanov@lp-0592:~/study/test_git$ git cat-file -p c464e7bb7d2ef08d74957199c2e367abd1bbb8cc
object cb8476f903326070640a4770171c1f76f9fbff27
type commit
tag v1.0
tagger Saltanov Kirill <saltanovlive@gmail.com> 1668974092 +0400

my version 1.0
```

Notice that the “object” the tag points to, via its hash, is the commit object, as we were expecting.

Refs

- Human-friendly link to an object
  - .git/HEAD
  - .git/refs/
- Types:
  - Heads
  - Tags
  - Remotes
  - Stash

```
saltanov@lp-0592:~/study/test_git/.git$ cat HEAD
ref: refs/heads/master
```

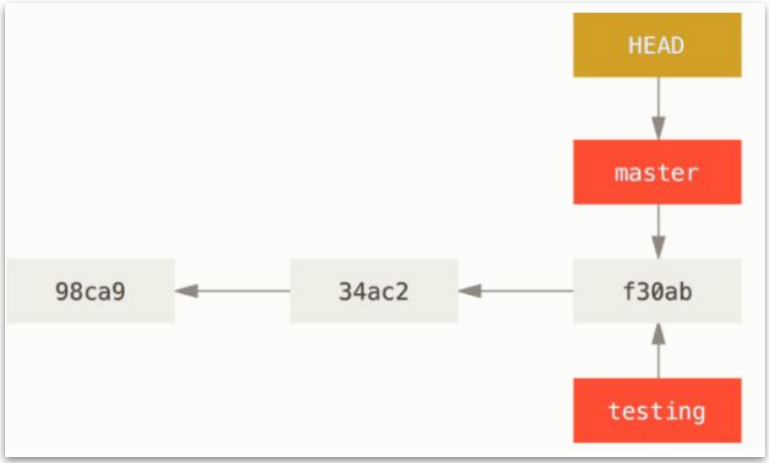
```
saltanov@lp-0592:~/study/test_git/.git/refs/heads$ cat master
cb8476f903326070640a4770171c1f76f9fbff27
```

```
saltanov@lp-0592:~/study/test_git/.git/refs/tags$ cat v1.0
c464e7bb7d2ef08d74957199c2e367abd1bbb8cc
```

Branch - is a moving pointer to some commit. Each time you commit, the pointer moves forward to the last commit.

Tag - is a static pointer to some commit. It's designed to be static and may be moved only forcefully (bad practice)

When creating a new branch



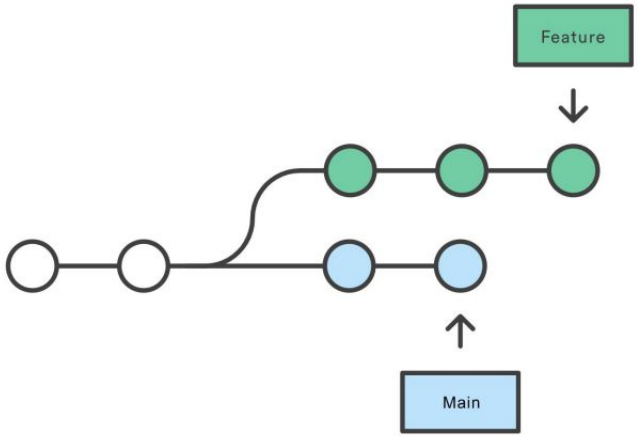
e.g. created a new branch testing

e.g. did new changes on the branch testing and master - divergent history --> both changes can be merged into one branch, so then HEAD will point again on the same commit

Working with branches

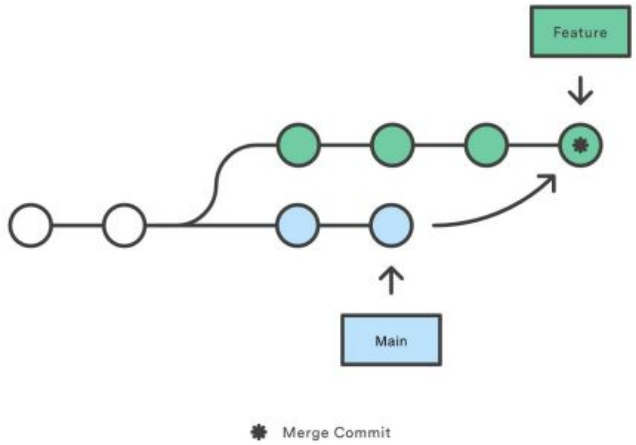
- `$git log [<SHA-1>]` # list history starting from HEAD or <SHA-1>
- `$git branch <branch_name>` # create a branch
- `$git branch [-d|-D]` # delete branch
- `$git branch` # list branches, with **[-v]** shows the short hash and commit message where branches point to
- `$git checkout <branch_name>` # switch to another branch
- `$git checkout -b <branch_name>` # create branch and switch to it
- `$git tag` #list created tags
- `$git tag -a <tag_name> [-m <MESSAGE>] [<SHA-1>]` # annotated tag

Ways to integrate changes back?



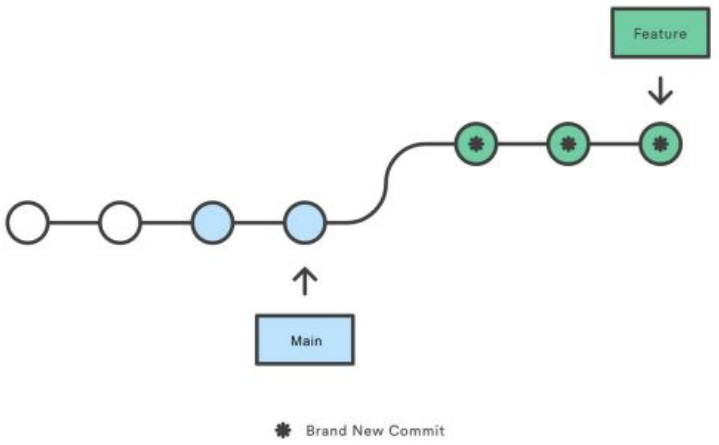
Ways to integrate changes back

Merge



- Merge keep all commits with all history

Rebase

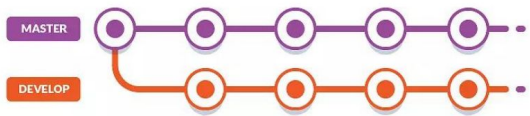


- Rebase discard all history and create a new commit when rebasing

Branching strategies

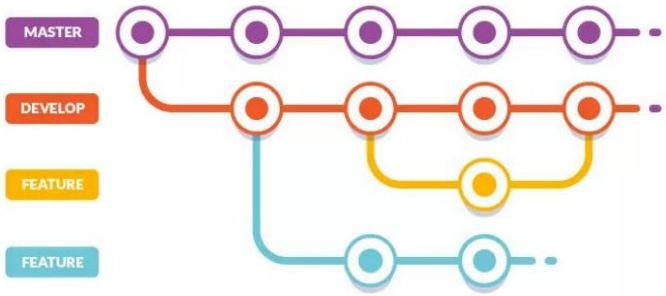
- Zero branch strategy

- Develop Branch Strategy

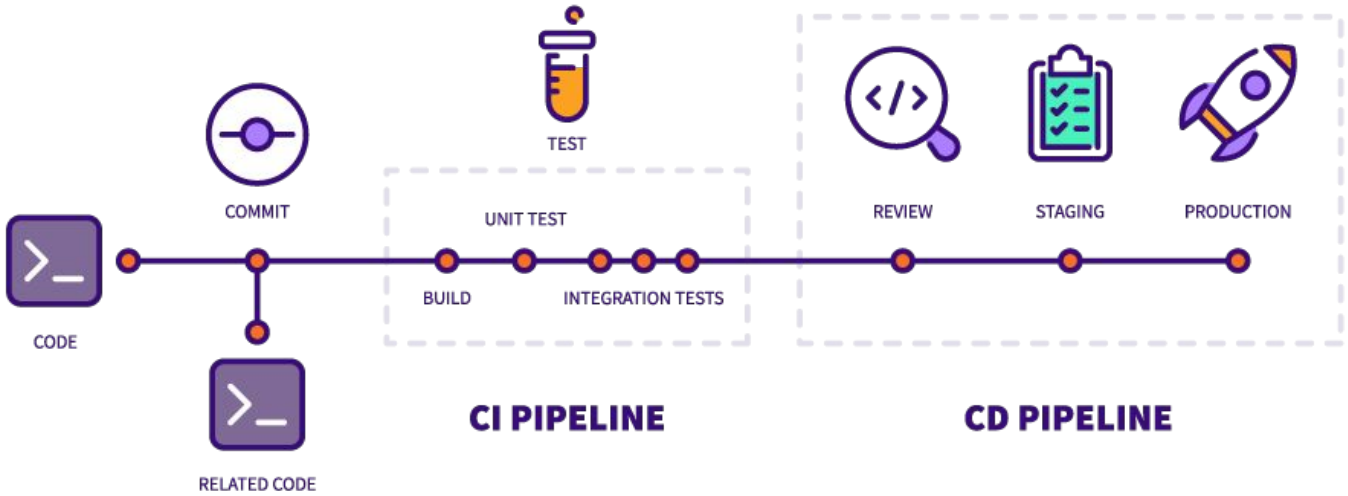


- Feature Branch Strategy

- GitFlow Branch Strategy



What is CI/CD


TEST
COMMIT
UNIT TEST
REVIEW
STAGING
PRODUCTION
BUILD
INTEGRATION TESTS
CODE
RELATED CODE
CI PIPELINE
CD PIPELINE

CI/CD with the Gitlab