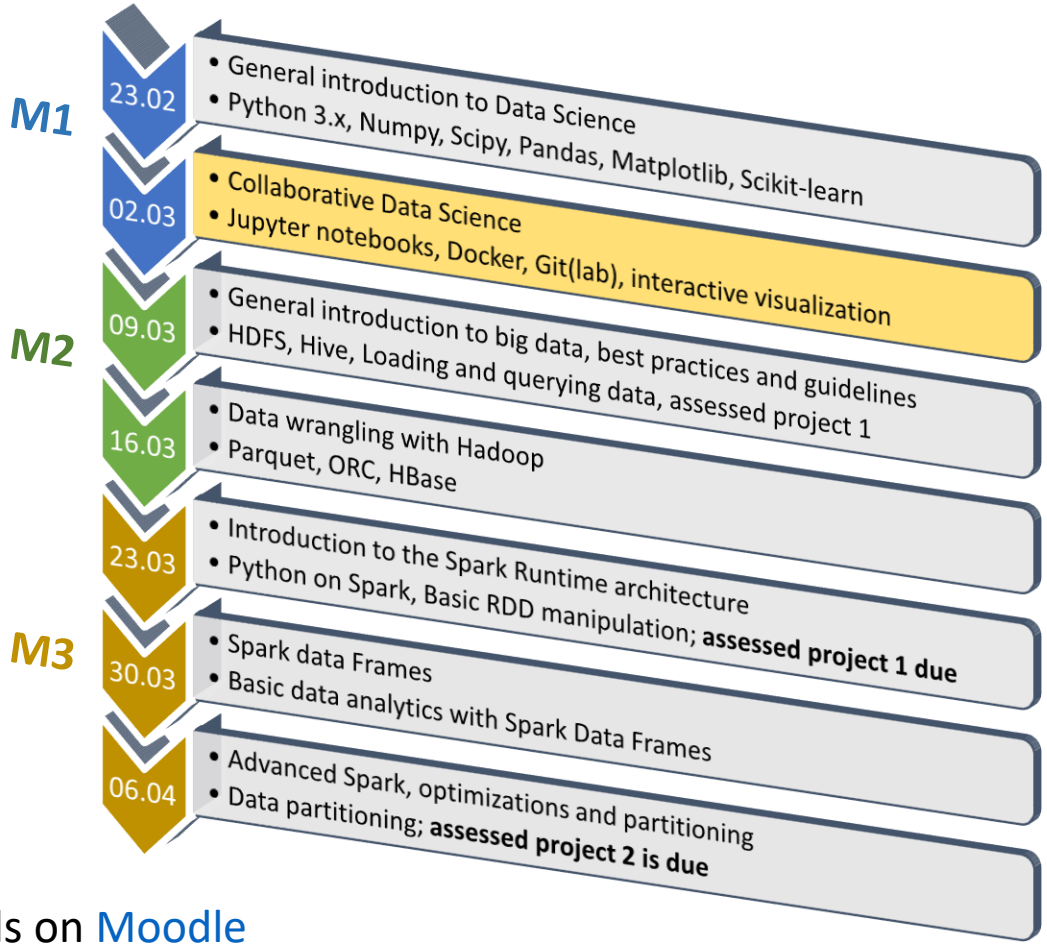
THE DATA SCIENCE LAB
Elements of Collaborative Data Science

COM 490 – Spring 2022

Week 2

Agenda Spring 2022



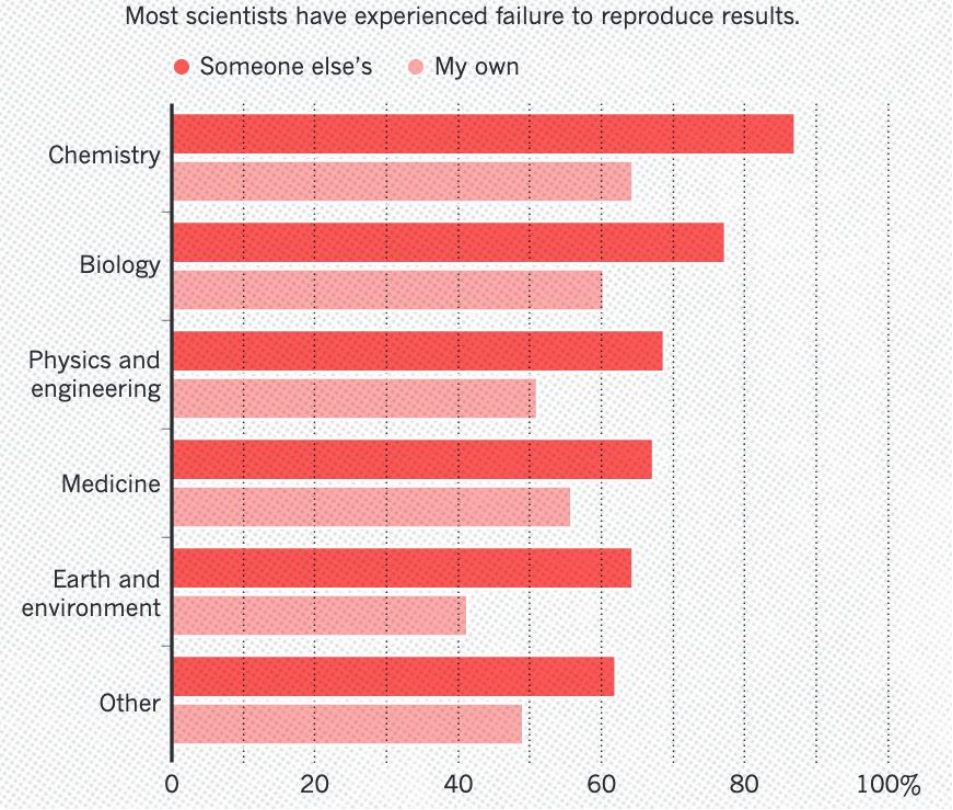
*Details on Moodle

Week 1 – Questions?

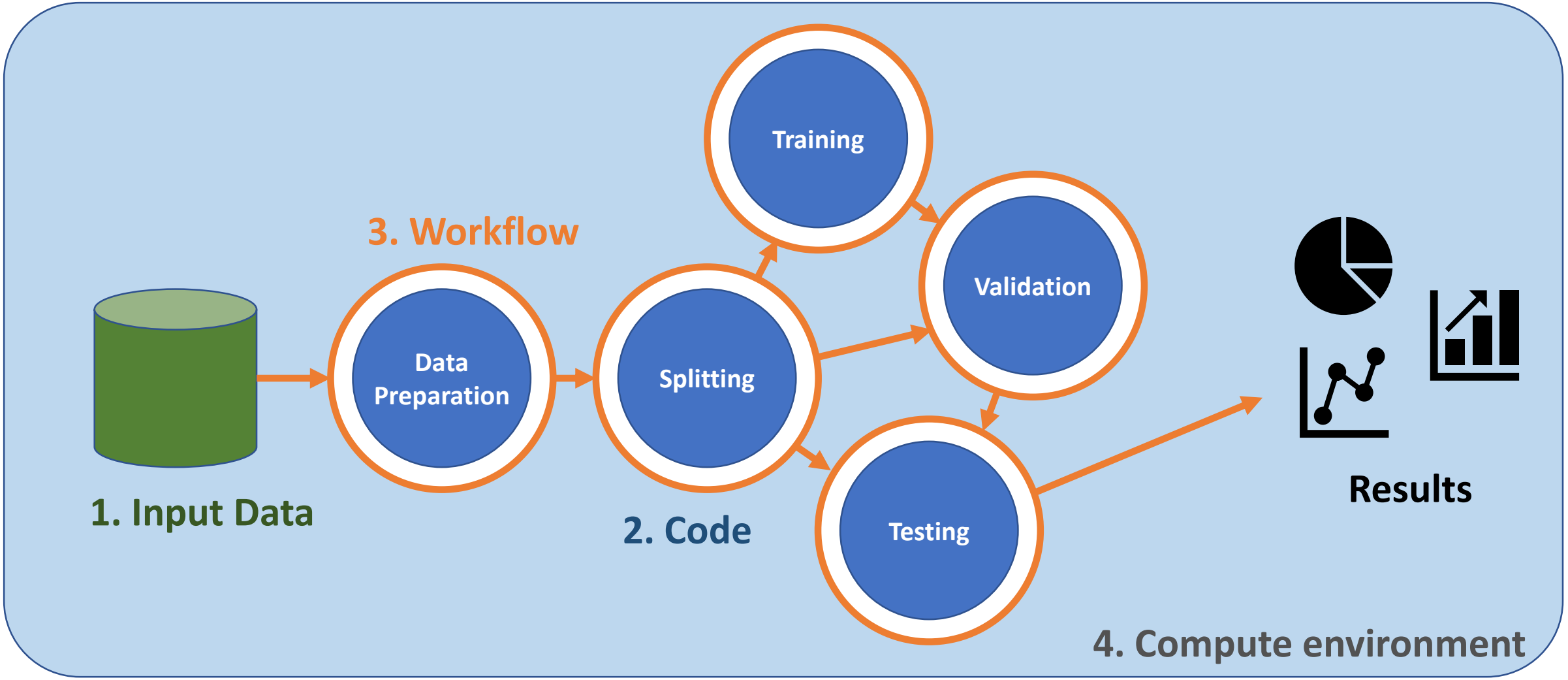
Forewords – Scientific Reproducibility “Crisis”

- It is real and cannot be ignored
  - [1,500 scientists lift the lid on reproducibility]

    (Nature, Aug. 2016)



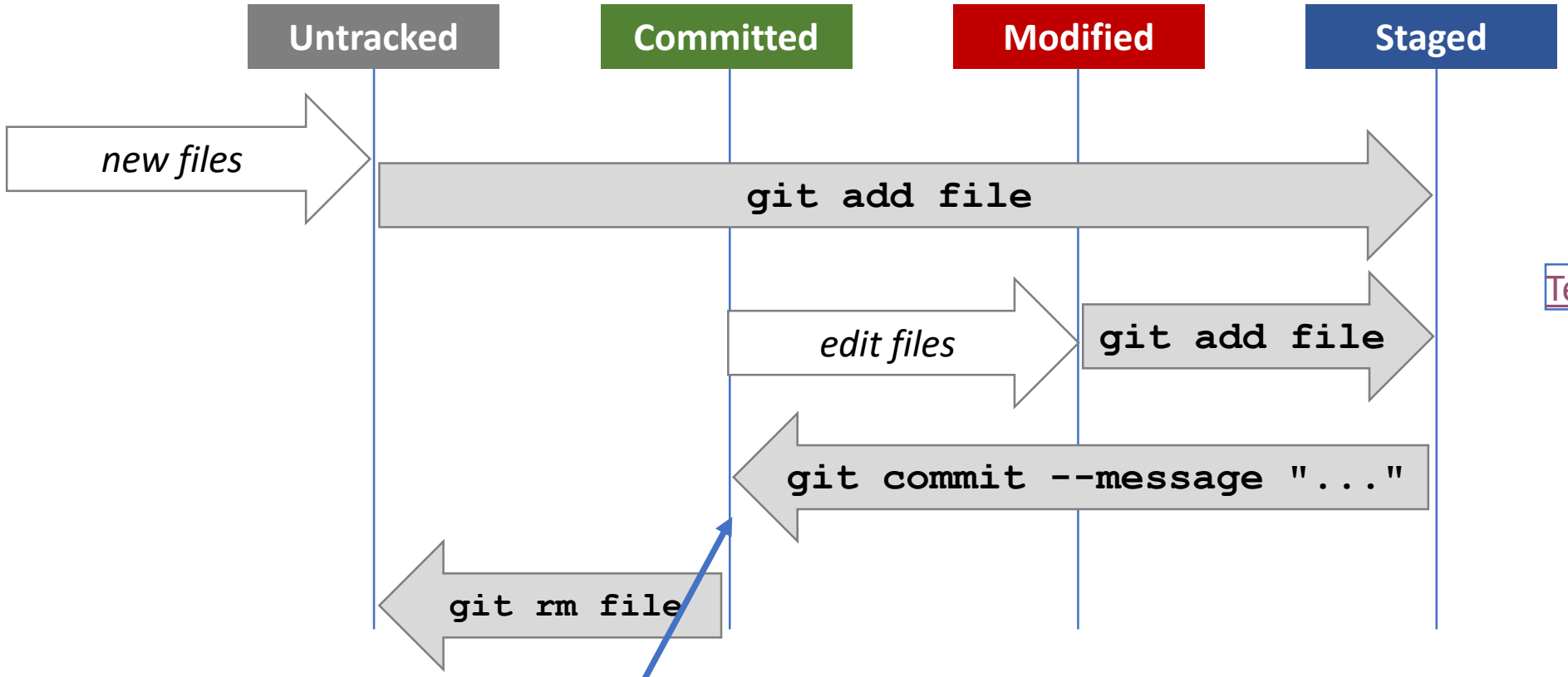
Ingredient of Collaborative Data Science



Collaborative Data Science - Tools

- **Code versioning**
  - Version Control Software: git, github or gitlab
  - *Alternatives: SVN, Mercurial, Bazaar, BitKeeper, CVS, RCS, ...*
- **Data versioning**
  - Large file versioning: git extension for large file systems (git lfs)
- **Compute environment**
  - Runtime environment
    - Docker images and containers
    - *Alternatives: LXC, VMWare, VirtualBox, ...*
  - Package managers (Python)
    - conda, pip
- **Workflow executor (not part of this course)**
  - Snakemake for Python, Apache Oozie for Hadoop
  - *Alternatives: make (and clones), nextflow, Cromwell/Toil (CWL), ...*

Git – File Lifecycle

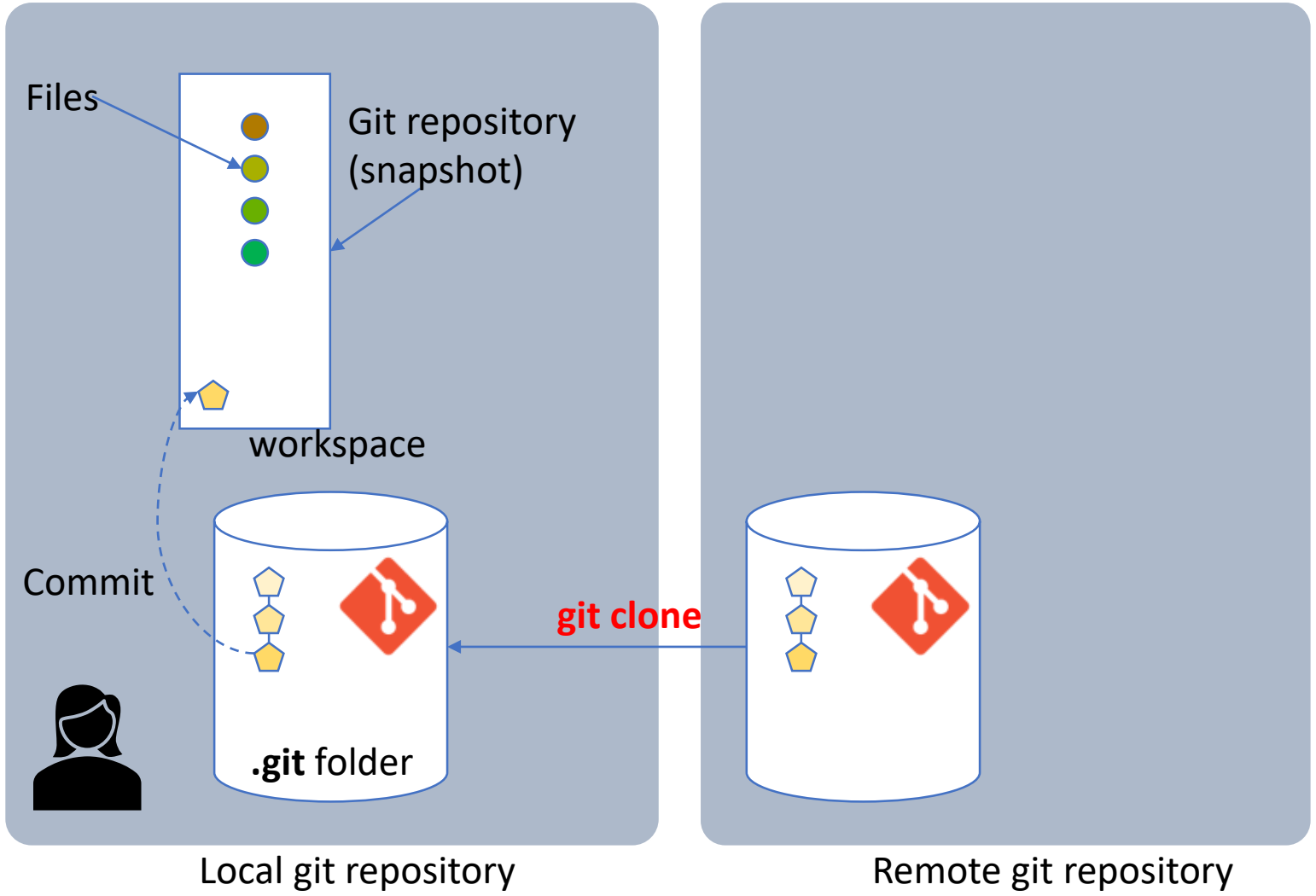


Terminal demo

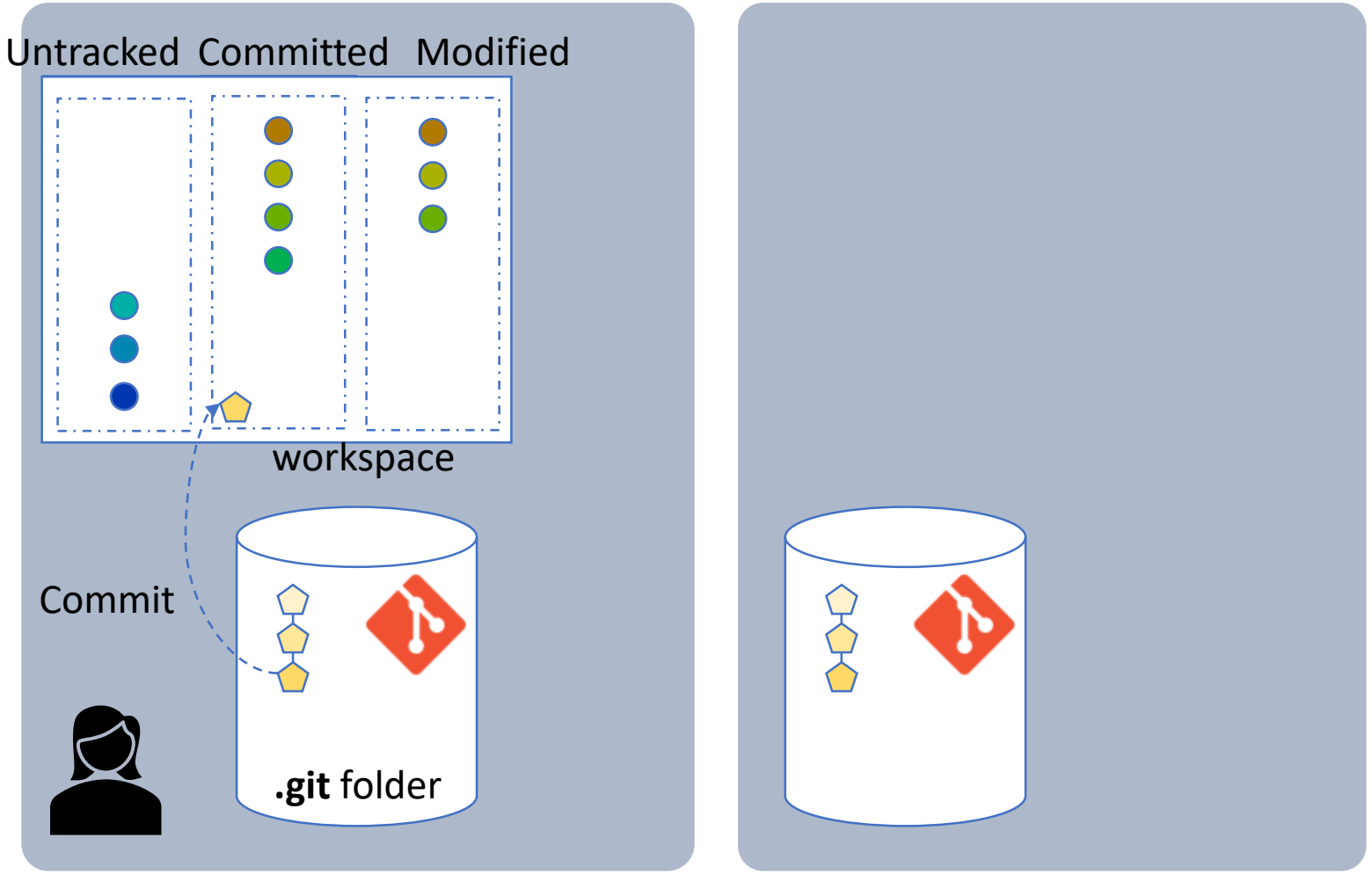
a new commit is created in the project repository

source: git-scm.com

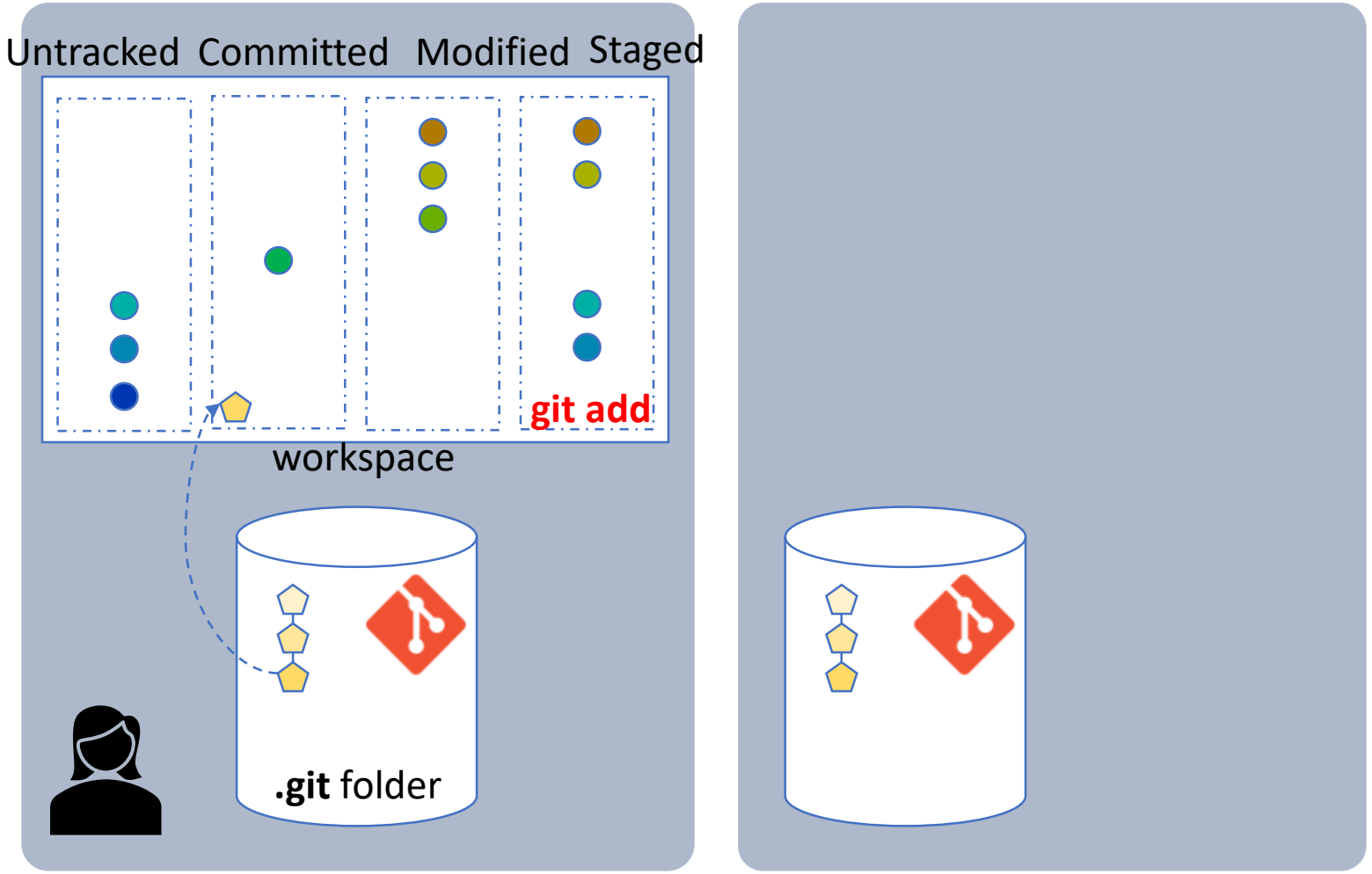
Git – Decentralized VCS



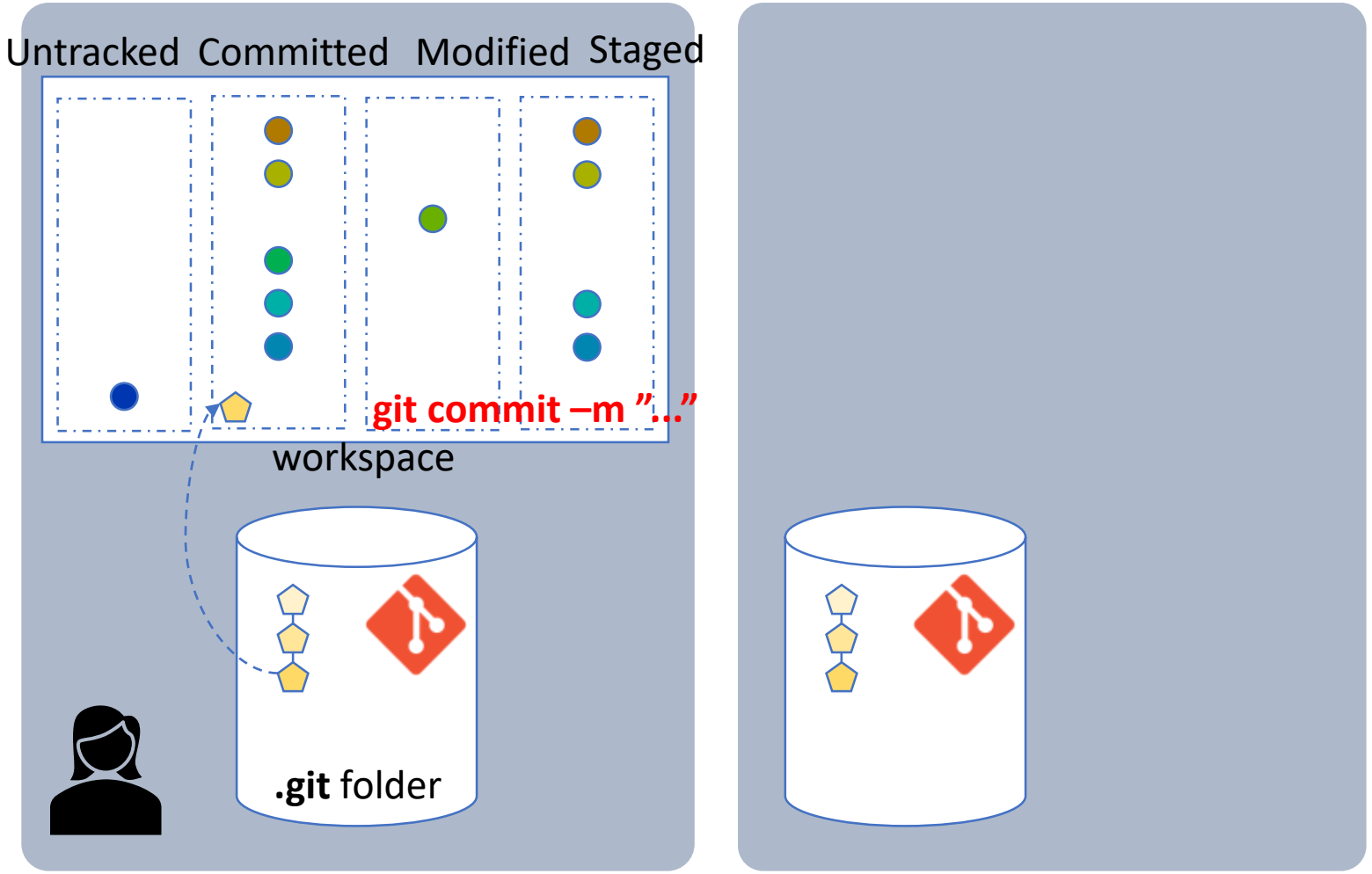
Git – Decentralized VCS



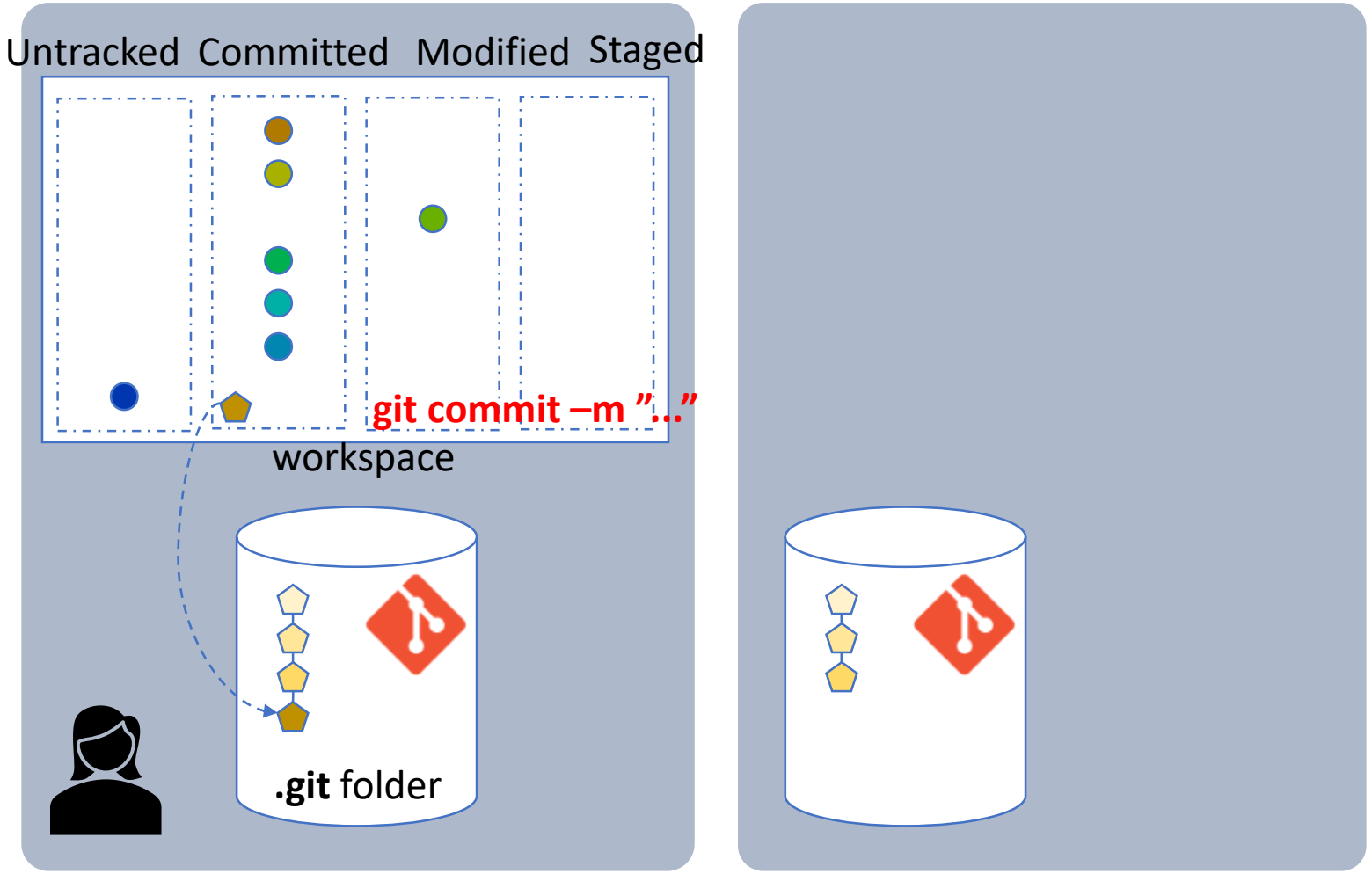
Git – Decentralized VCS



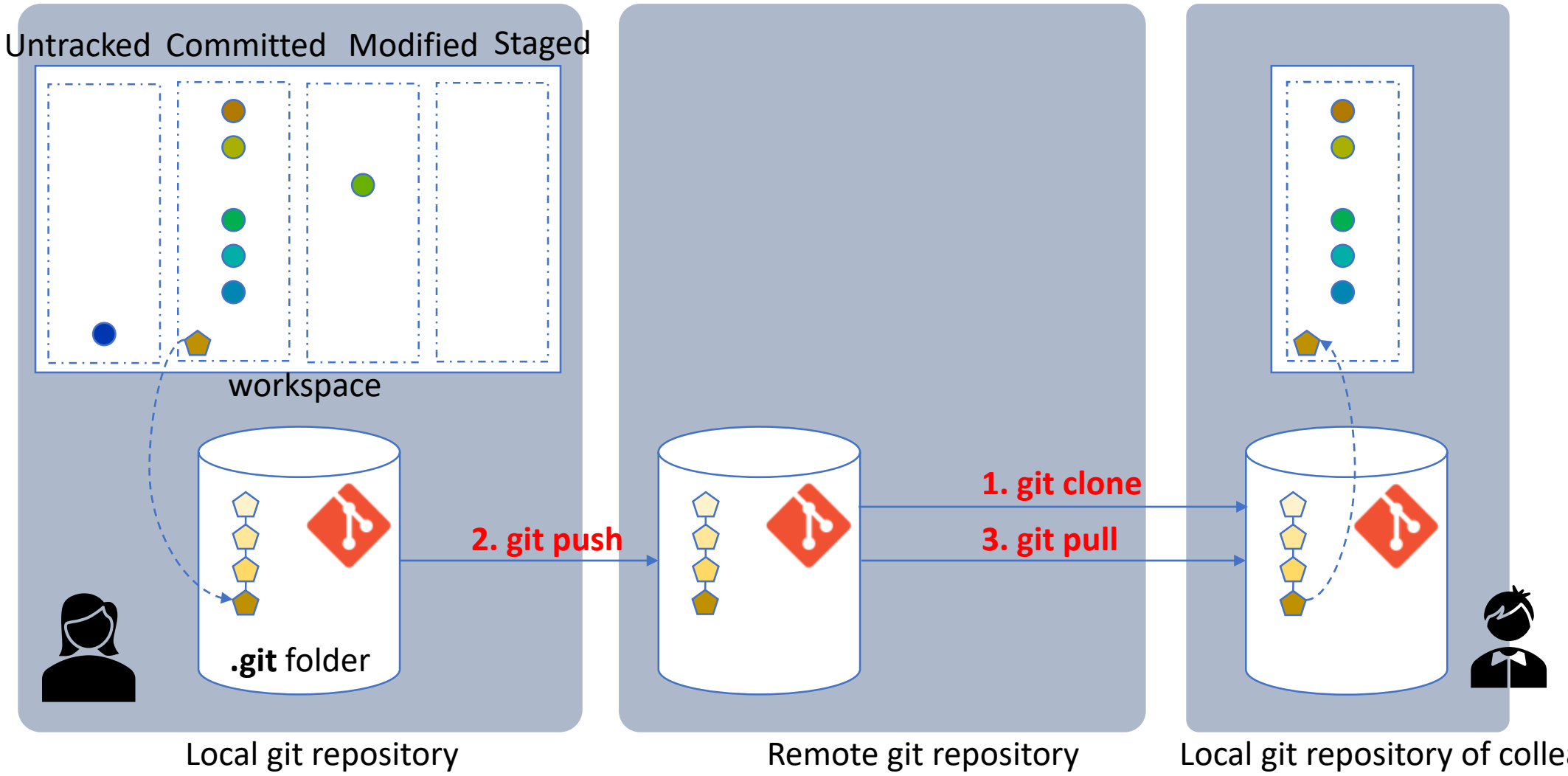
Git – Decentralized VCS



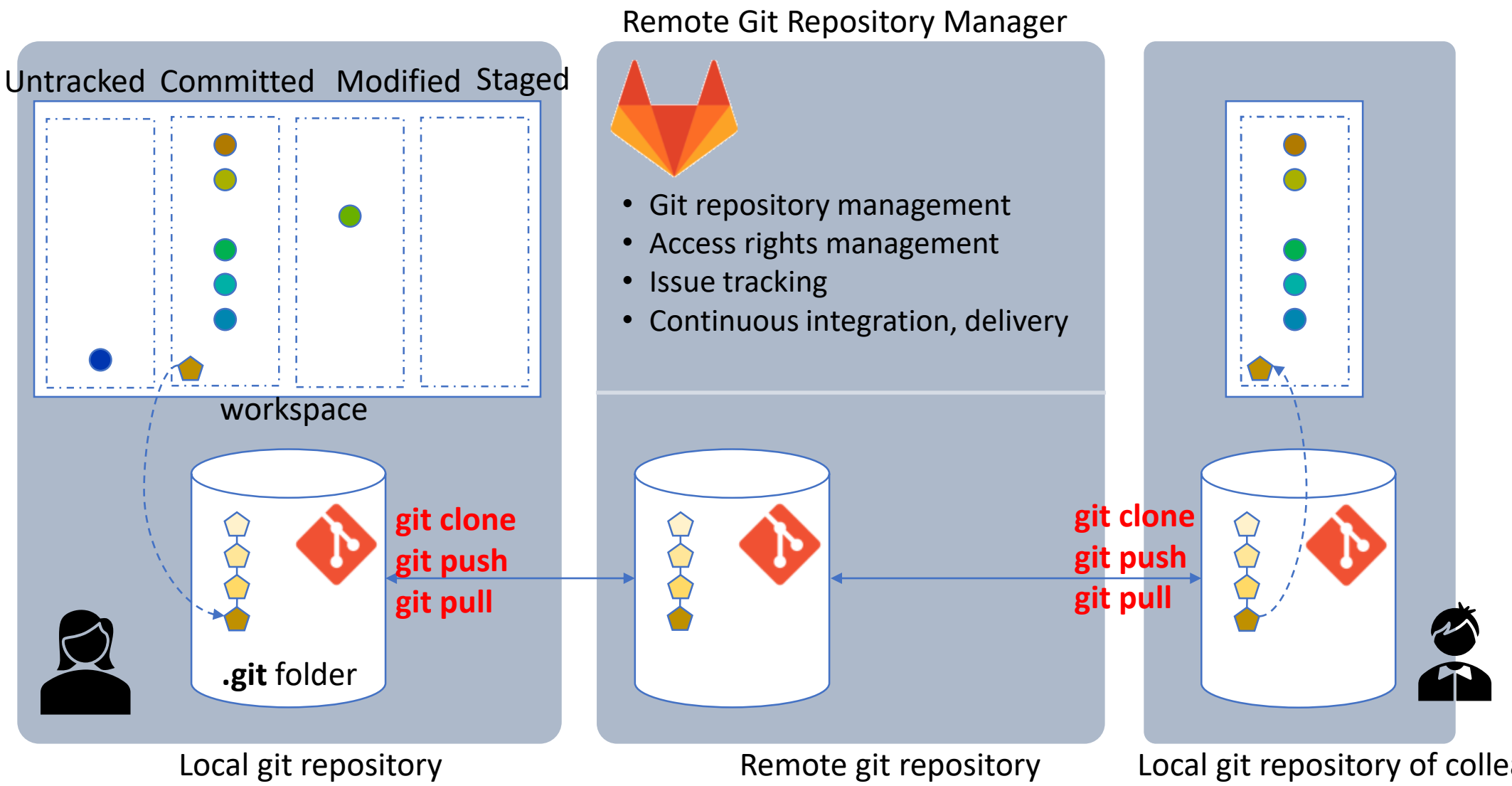
Git – Decentralized VCS



Git – Decentralized VCS

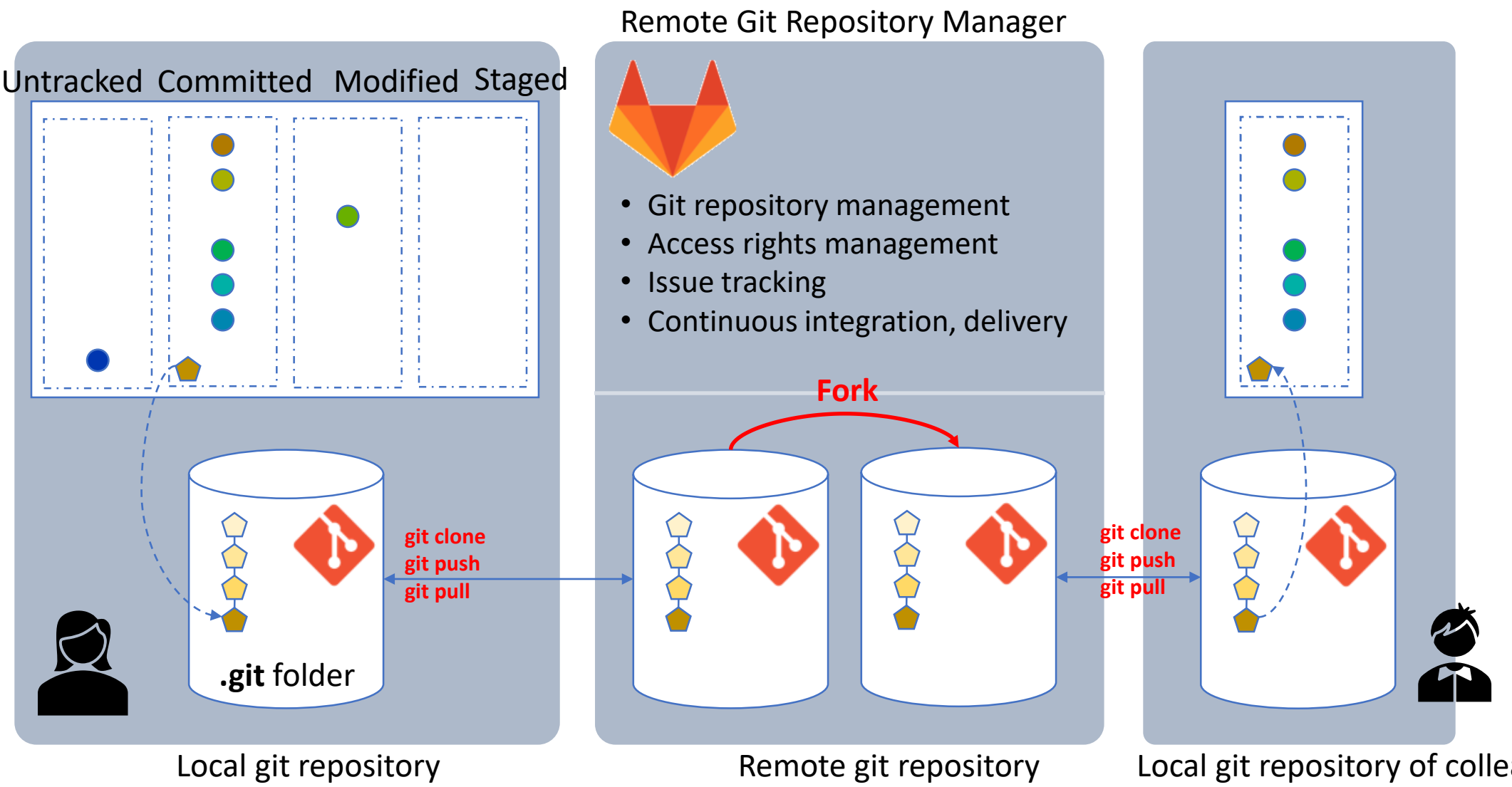


Git – Decentralized VCS

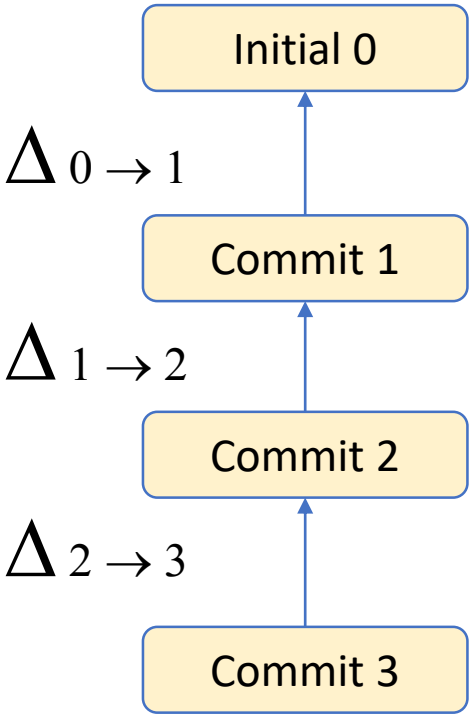


- Git repository management
- Access rights management
- Issue tracking
- Continuous integration, delivery

Git – Decentralized VCS



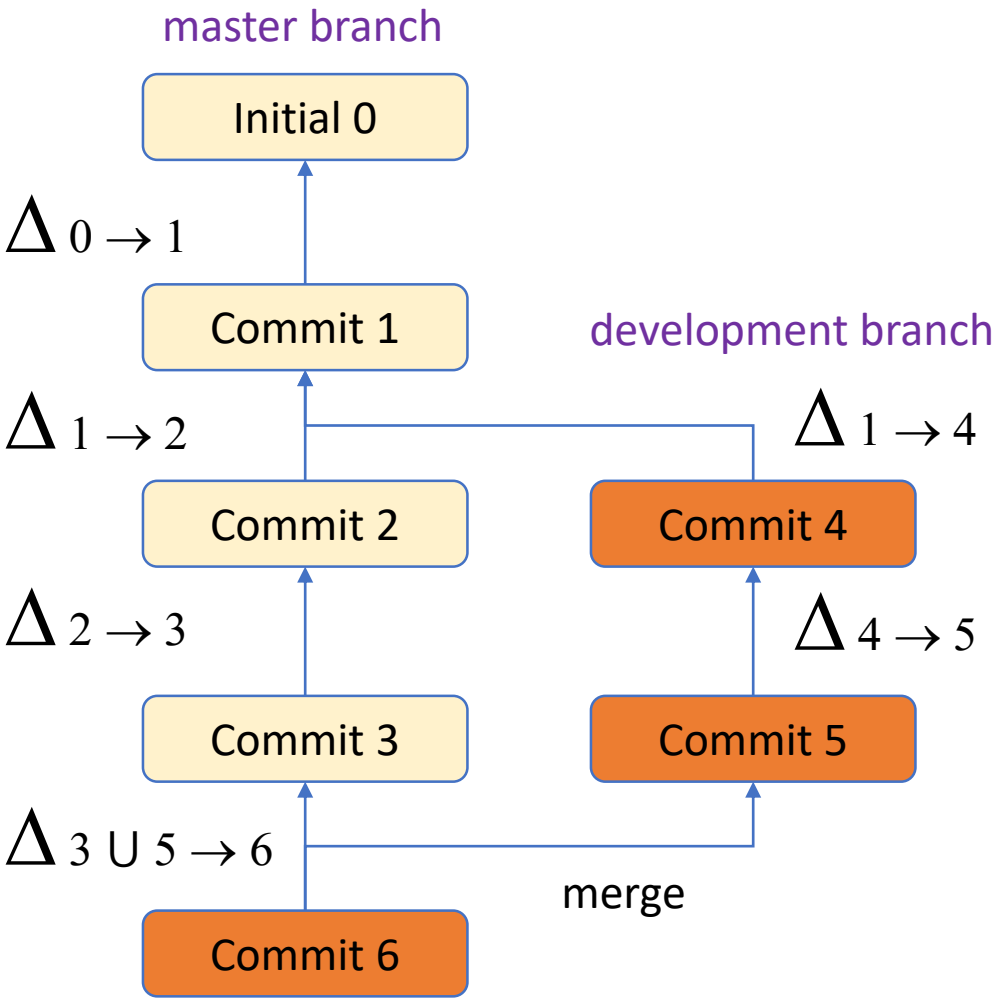
Git – Commit History (git log)



- A commit is a single point in the commit history. It is a snapshot of all the tracked files at that point.
- By default, a succession of commits follow a linear evolution

source: git-scm.com

Git – Branching



- A commit is a single point in the commit history. It is a snapshot of all the tracked files at that point.
- By default, a succession of commits follow a linear evolution
- Using Git, it is also possible to work in parallel on separate branches.

Git branching strategies

- GitFlow (complex)
- Github Flow (easy)
- Gitlab Flow (easy)
- OneFlow (medium)
- …

source: git-scm.com

Git – Common Commands

- Create a new repository

```
git init
```

- Or, copy a repository locally from a remote origin – linked to origin (pull/push)

```
git clone https://dslab2021-renku.epfl.ch/gitlab/com490-pub/w2-exercises.git
```

- Verify state of repository (untracked, modified, staged files, commits ahead)

```
git status
```

- Stage files for a grouped commit

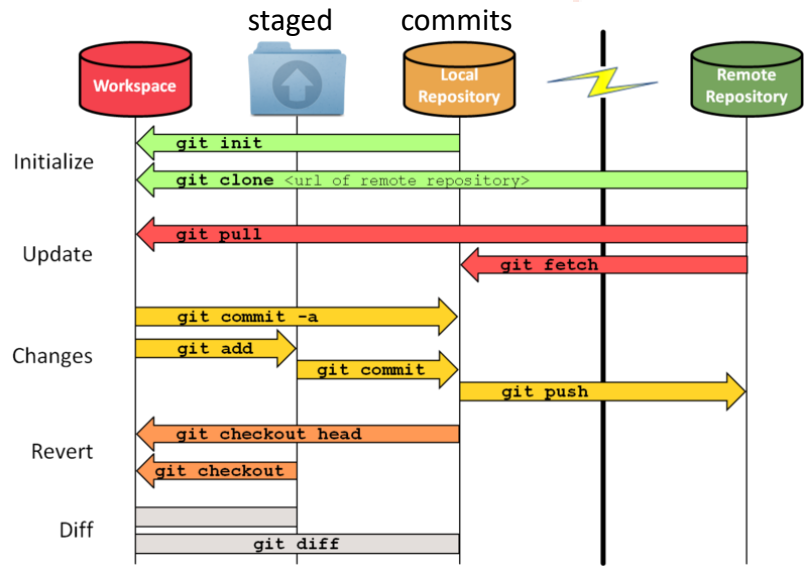
```
git add files+
```

```
git add .
```

- Commit files from staged area

```
git commit --message "description courte de la validation"
```

Git – Common Commands



- Display commit log (project history)

```
git log [--all][--graph]
```

- Checkout earlier commit

```
git checkout {commit-hash|branch-name}
```

- Propagate new commits to remote (origin) repository

```
git push
```

- Retrieve latest commits from remote (origin) repository

```
git pull
```

- Get help

```
git command --help
```

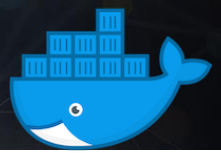
Git LFS - Large File Systems

- Git LFS is used to work with big files (e.g. data files)
  - Replace large files with tiny text files inside git (hash of file content), pointing to content stored outside git
  - Pointer files are swapped out for the real files at checkout (using git lfs smudge and clean)
- Basic commands
  - git lfs track ***files+***
  - git lfs ls-files
  - git lfs fetch
- Most common mistakes
  - You believe you are reading the real file, but it is a pointer file (e.g. you forgot to do a git lfs fetch)
  - You have an existing file, track it with git lfs and replace it with a tiny pointer file, but the large file is still in the git history
- More advanced commands
  - git lfs migrate # convert history of git repository to use Git LFS (when large files are accidentally committed)
- Refs:
  - https://git-lfs.github.com
  - https://github.com/git-lfs/git-lfs/wiki/Tutorial

Runtime Environment – Containerization

- Containerization of applications: isolation and portability
  - "It's all about encapsulating or packaging the software code and all its dependencies so that it can run consistently and coherently on any infrastructure. ”
  - Popular containerization systems: docker (LXC, ...)

Docker containers



- Docker
  - Used by developers to build, share and run software in areas called containers
- Some Docker concepts
  - **Docker image**: a binary containing the executables and dependencies of the software.
  - **Docker container**: an instance of the image that resides in memory and uses the CPU of the host machine.
  - Containers are isolated from each other and from the machine for security reasons
  - A user can safely (in theory) be granted admin right inside a container, without being admin on the host machine

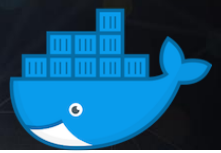
Dockerfile - Demo

- Dockerfiles are ‘recipes’
  - Provide instructions to build an image
  - Install software, include files, …
- Docker on Renku
  - Each project has a Dockerfile
  - Defines the environment of sessions
  - You can change it !

Dockerfile 1.97 KB

```
# For finding latest versions of the base image see
# https://github.com/SwissDataScienceCenter/renkulab-docker
ARG RENKU_BASE_IMAGE=renku/renkulab-py:3.9-0.10.1
FROM ${RENKU_BASE_IMAGE}

# Uncomment and adapt if code is to be included in the image
# COPY src /code/src

# Uncomment and adapt if your R or python packages require extra linux (ubuntu) software
# e.g. the following installs apt-utils and vim; each pkg on its own line, all lines
# except for the last end with backslash '\' to continue the RUN line
#
# USER root
# RUN apt-get update && \
#     apt-get install -y --no-install-recommends \
#     apt-utils \
#     vim
# USER ${NB_USER}

# install the python dependencies
COPY requirements.txt environment.yml /tmp/
RUN conda env update -q -f /tmp/environment.yml && \
    /opt/conda/bin/pip install -r /tmp/requirements.txt && \
    conda clean -y --all && \
    conda env export -n "root"

# RENKU_VERSION determines the version of the renku CLI
# that will be used in this image. To find the latest version,
# visit https://pypi.org/project/renku/#history.
ARG RENKU_VERSION=0.16.2

######################################################
# Do not edit this section and do not add anything below
```

Docker – Common Dockerfile Directives



Directive	Description
FROM alpine:latest	Base image. Have a look at https://hub.docker.com/explore/
USER root	Set the user for the following commands
RUN apt-get update && apt-get install –y renku	Run arbitrary shell commands **Each RUN adds an overlay layer**!
COPY ./requirements.txt /tmp/requirements.txt	Put a copy of a local file into the docker image**, it also add a new layer**!
EXPOSE 80 443	Expose a port (also -p port1:port2 of docker run)
VOLUME ["/var/log"]	Specify mount points for external volumes, i.e. expose external volumes inside docker container (see also –v or --mount of docker run)
ENTRYPOINT ["/bin/bash”,”-i”]	The command that is run when container is started (see also **CMD**)

Common mistakes:

- Each directive must be on one line, use \ to break lines
- Do not assume bash for the shell (it’s a simple shell)
- Directives add up. If a file is added in a layer and is removed in the next – it becomes hidden, and is still using space.

References

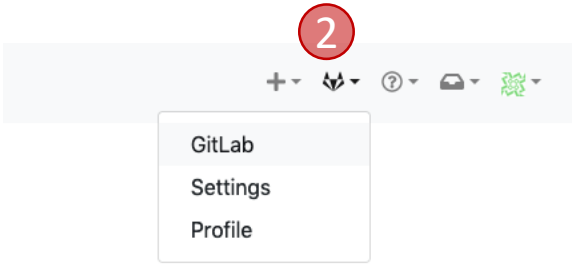
- Git, Gitlab
  - git documentation
  - git lfs documentation
  - git lfs tutorial
  - gitlab basics
- Docker
  - Documentation
- Conda
  - Command reference

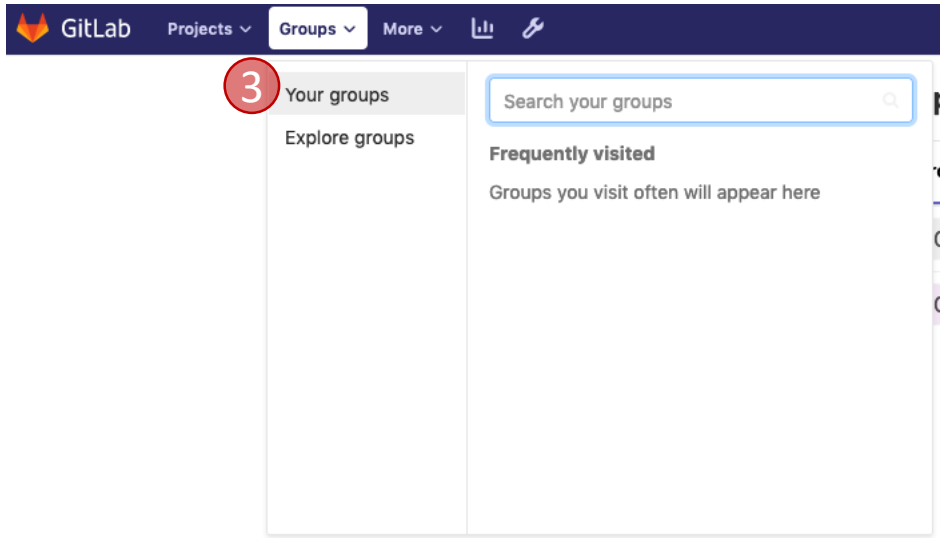
Start your engines

https://dslab2022-renku.epfl.ch/projects/com490/lab-course

RENKU – Creating a Gitlab group

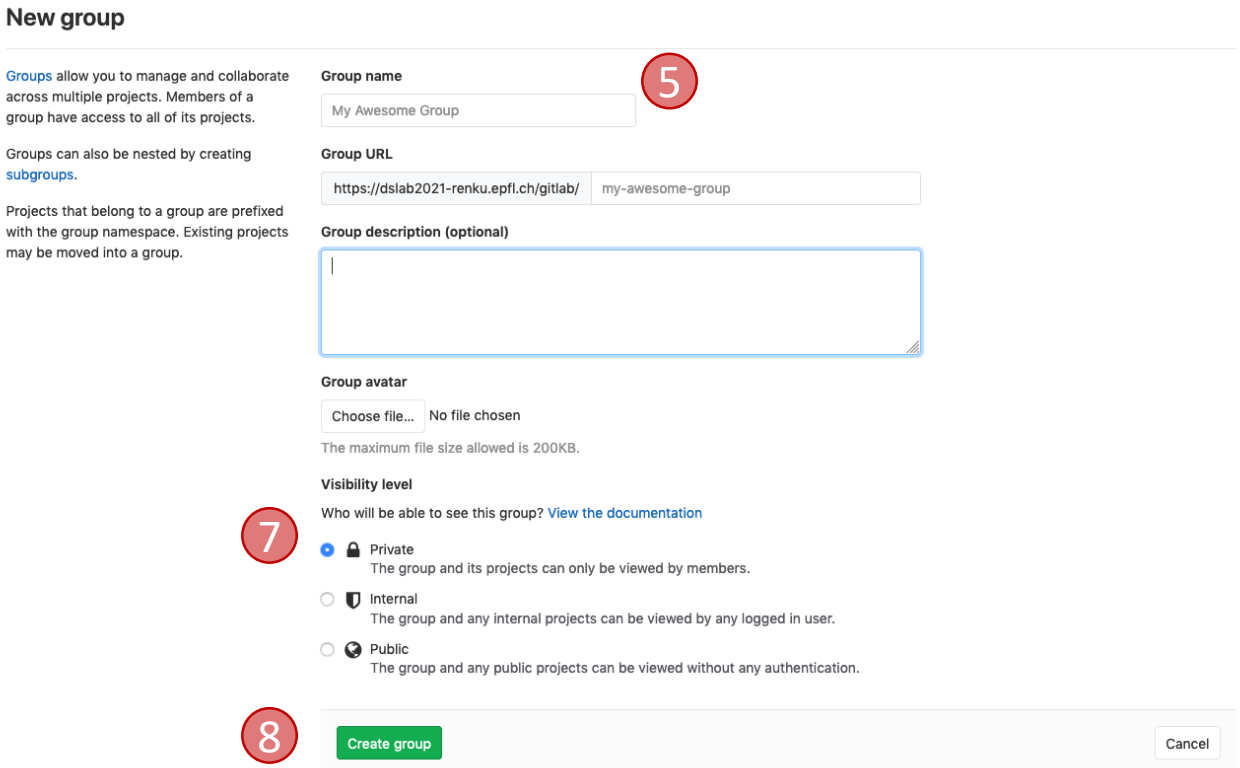
1. Login on https://dslab2022-renku.epfl.ch
2. Open the gitlab view (top right corner)
3. Under Groups select Your Groups or Explore Groups
4. Select New group (top right)





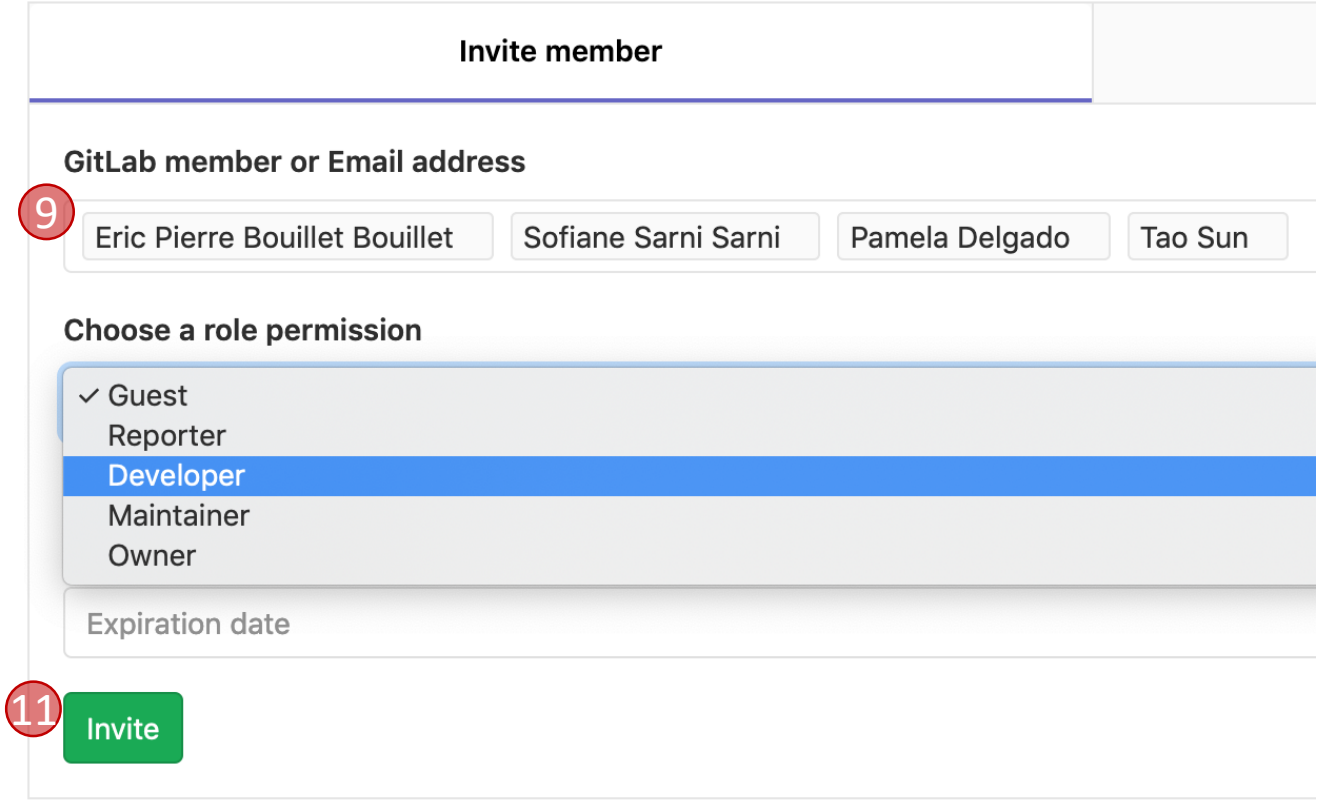
RENKU – Creating a Gitlab group

5. In New Group enter a name for your group
6. You can leave the description empty
7. Make sure your group is private
8. Click Create group.



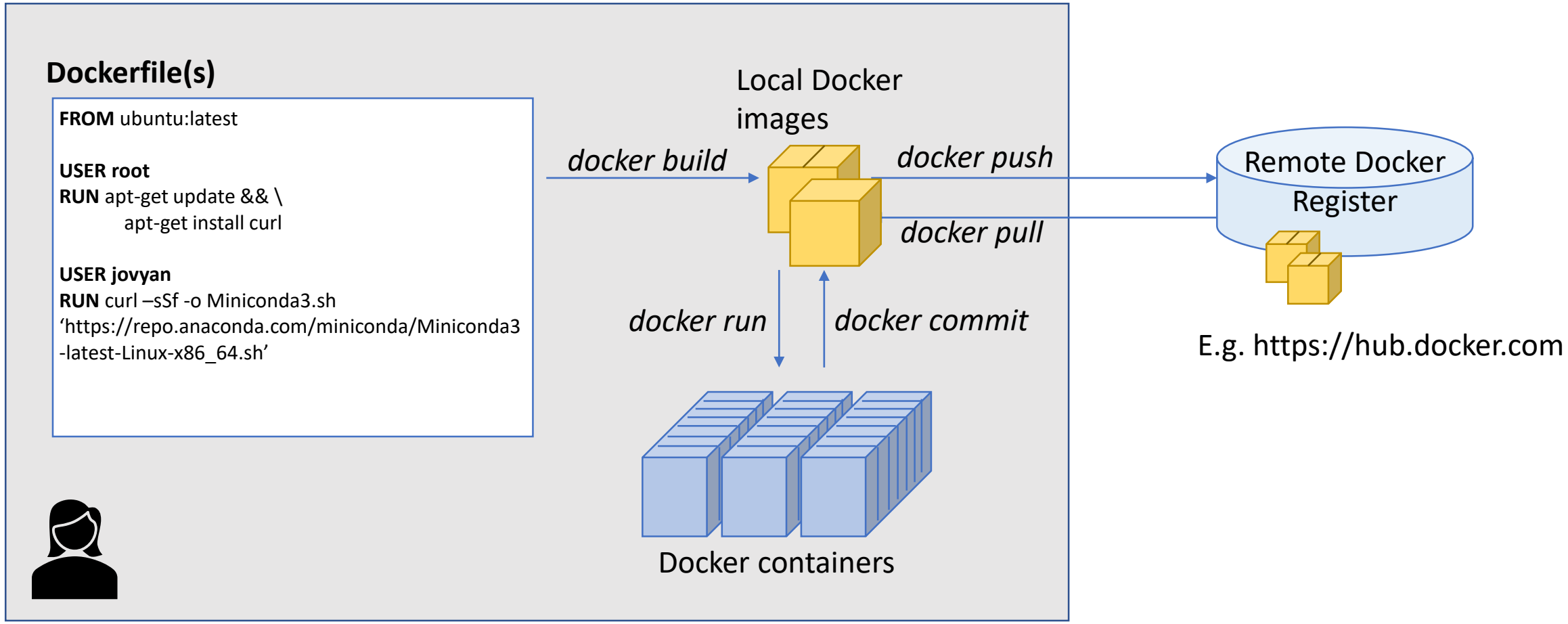
RENKU – Creating a Gitlab group

9. In the newly created group, list your collaborators under “Invite member” tab
10. Give your collaborators, the Developer role permission at a minimum
11. **Invite** .
12. You can later add/remove/edit the members of the group.
13. Remember to keep the group private!
14. See [gitlab groups] documentation



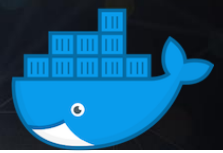
Additional slides

Docker - Overview



Docker Engine running on server

Docker – Common Commands



- Create a Docker instance from an image, download image if not available locally

```
docker run [-it][--rm][-p port:port] [docker-image[:version]] [command]
```

- List docker instances

```
docker ps --all
```

- Stop and start docker instance

```
docker stop instance-id
```

```
docker start instance-id
```

- Kill and remove instance

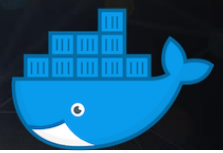
```
docker kill instance-id
```

```
docker rm instance-id
```

- Attach (stdin, stdout, stderr) to a running instance

```
docker attach instance-id
```

Docker – Common Commands



- Run command inside a running Docker instance (possibly with extended privileges)

```
docker exec [-it][--privileged][--user uid:gid] instance-id command
```

- Build image from a *Dockerfile*.

```
docker build .
```

- List images cached locally

```
docker images
```

- Remove local image

```
docker rmi instance-id
```

- Get help for other commands and additional arguments

```
docker [command] --help
```