《Rust程序设计语言》课程项目

基于Rust的Git系统

授课教师：冯洋

项目助教：燕言言

DG21320008@smail.nju.edu.cn

南京大学计算机学院

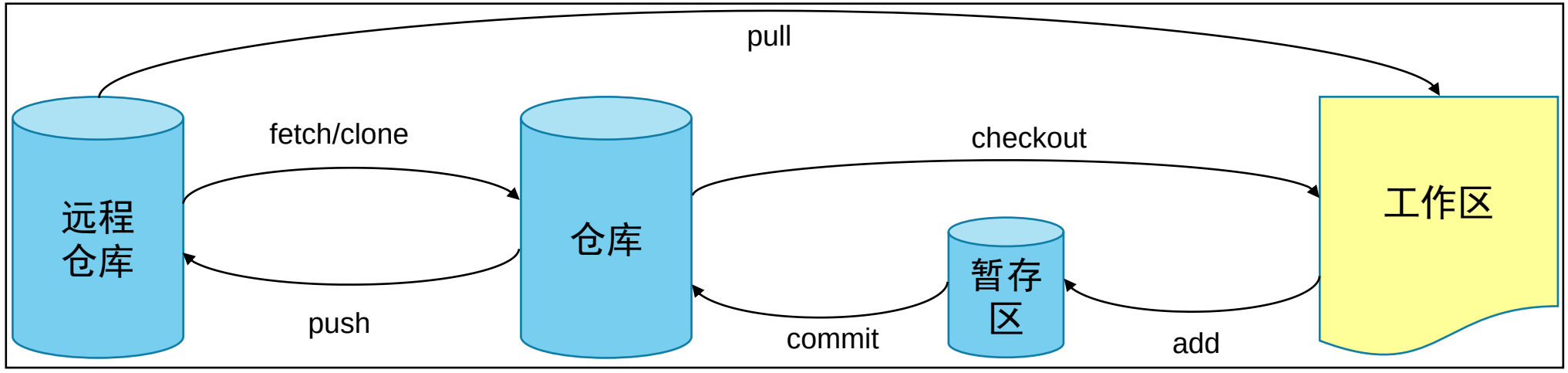
项目目标

- 本项目旨在使用 Rust 作为主要开发语言，实现一个简易的 Git 系统[1-2]。通过完成该项目，同学们将深入理解 Git 的核心原理，并掌握 Rust 语言在实际项目中的应用。
- 项目内容
  - 基本功能：支持 Git 的基本操作，包括 git init（初始化仓库）、git add（添加文件到暂存区）、git rm（删除文件）、git commit（提交更改）等基础功能。
  - 分支管理：实现分支管理功能，如 git branch（创建/删除分支）、git checkout（切换分支）以及 git merge（合并分支）。
  - 进阶功能：鼓励有兴趣的同学进一步实现远程操作功能，例如 git fetch、git pull 和 git push，以模拟完整的 Git 工作流程。

[1]. Rust实现Git系统. Online Available. [2025]. https://github.com/GitoxideLabs/gitoxide
[2]. Java实现Git系统. Online Available. [2025]. https://sp21.datastructur.es/materials/proj/proj2/proj2

项目需求说明

- Git简介
  - Git 是一个分布式版本控制系统（DVCS）[1]，广泛用于软件开发中的版本控制和协作。它由 Linus Torvalds 在 2005 年创建。
  - 最初是为了管理 Linux 内核开发而设计的。Git 的核心思想是跟踪文件的变化，并允许开发者在不同的分支上并行工作，最终将这些变化合并到一起。

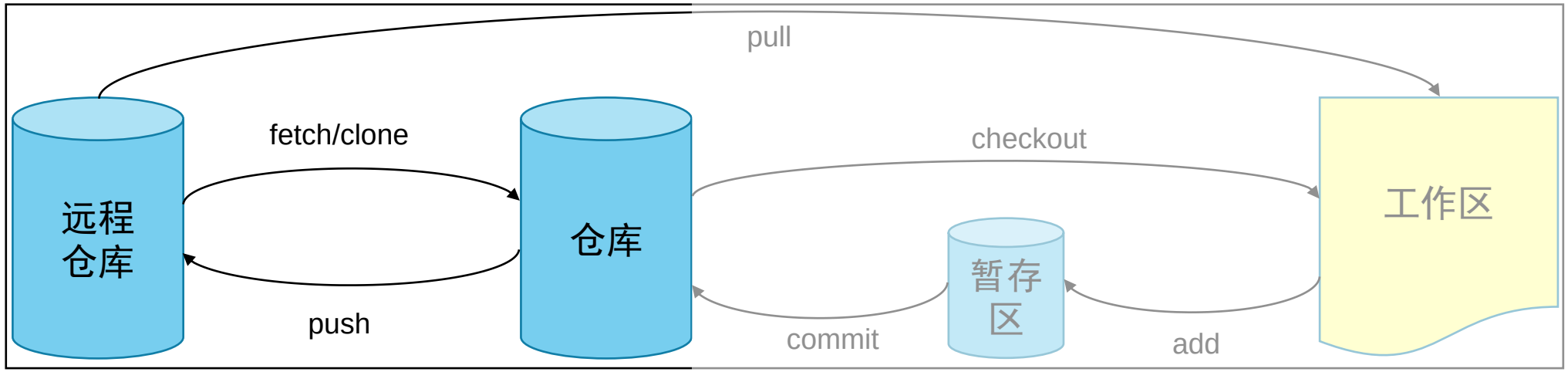


Git工作原理

[1]. Git原理与使用介绍. Online Available. [2025]. https://missing.csail.mit.edu/2020/version-control/

项目需求说明

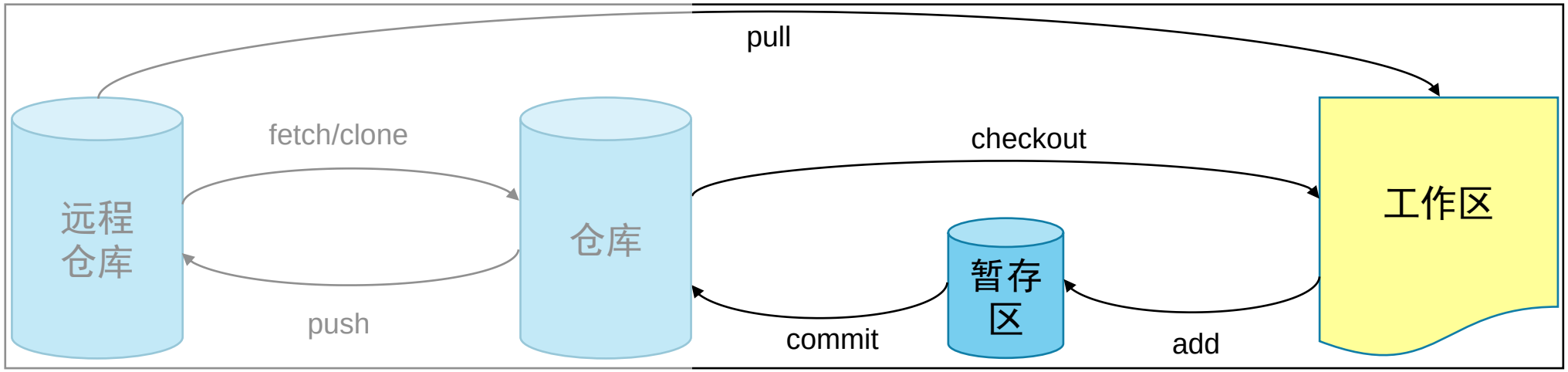
- 核心概念
  - Git 的设计基于一些核心概念，理解这些概念是掌握 Git 的关键。
    - 仓库：Git 仓库是 Git 用来存储项目历史记录的地方。它包含项目的所有文件、目录以及这些文件的变更历史。仓库分为两种：
      - 本地仓库：存储在开发者本地机器上。
      - 远程仓库：存储在远程服务器上（如GitHub、GitLab等），用于团队协作。



Git工作原理

项目需求说明

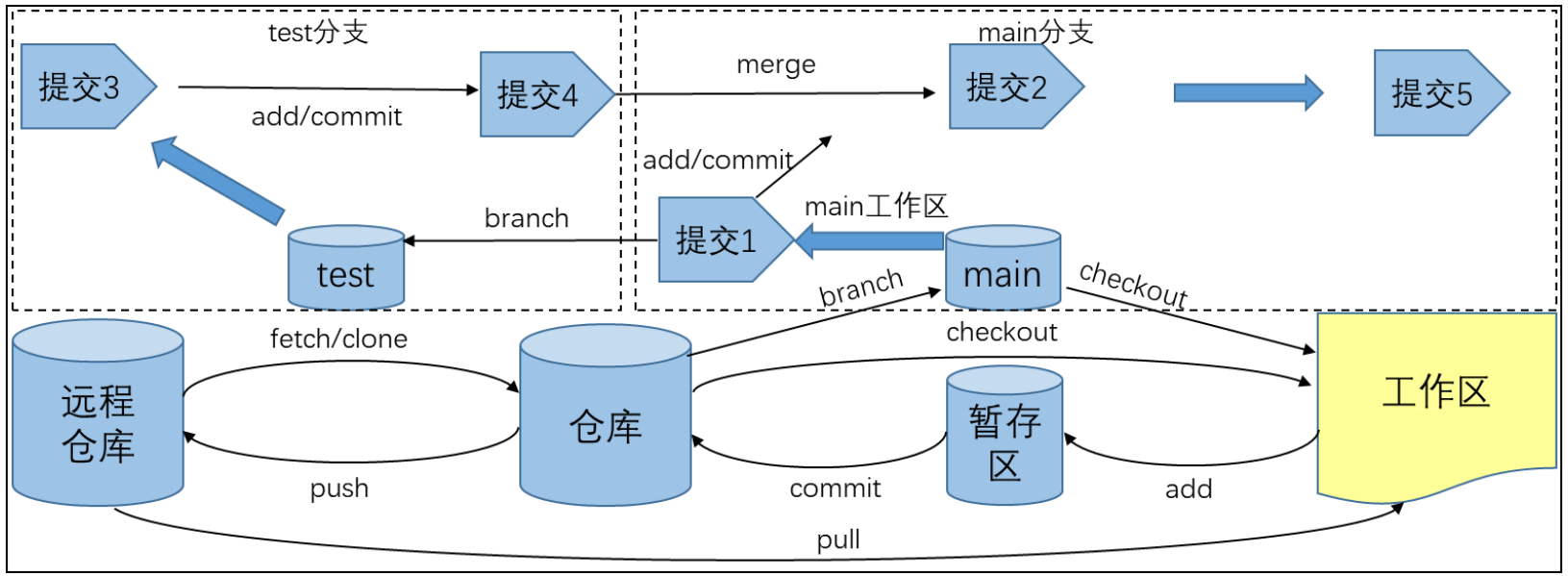
- 核心概念
  - Git 的设计基于一些核心概念，理解这些概念是掌握 Git 的关键。
    - 工作区：工作区是开发者当前正在编辑的文件和目录。它是从 Git 仓库中检出的文件副本。
    - 暂存区：暂存区是一个临时区域，用于保存即将提交的更改。开发者可以通过 git add 将文件添加到暂存区。
    - 提交：提交是 Git 中的一个快照，记录了某个时间点工作区和暂存区的状态。每次提交都会生成一个唯一的 SHA-1 哈希值，用于标识该提交。



Git工作原理

项目需求说明

- 核心概念
  - Git 的设计基于一些核心概念，理解这些概念是掌握 Git 的关键。
    - 分支：分支是 Git 的核心功能之一，允许开发者在独立的分支上开发新功能或修复问题，而不会影响主分支（通常是 main 或 master）。分支的本质是一个指向某个提交的指针。
    - 合并：合并是将两个分支的历史记录合并到一起的操作。Git 会自动尝试合并更改，但如果两个分支修改了同一部分代码，可能会产生冲突，需要手动解决。



Git分支合并

项目需求说明

- Git底层实现
  - Git 的底层实现基于以下几个关键机制：
    - 对象存储：Git使用四种对象来存储数据。
      - blob：存储文件内容。
      - tree：存储目录结构，指向 Blob 或其他 Tree。
      - commit：存储提交信息，指向一个 Tree 和父提交。
      - tag：存储标签信息，指向某个 Commit。
    - SHA-1 哈希：Git 使用 SHA-1 哈希算法为每个对象生成唯一的标识符。这个哈希值用于标识对象的内容。
    - 引用：引用是指向 Commit 的指针，例如分支（Branch）和标签（Tag）。
      - .git/refs 目录存储了所有的引用。
    - HEAD：HEAD 是一个特殊的引用，指向当前分支的最新提交。

项目需求说明

- Git应用场景与生态
  - ➢典型的应用场景：
  - 版本控制：跟踪代码的历史变更。
  - 团队协作：多人协作开发同一个项目。
  - 代码审查：通过 Pull Request 进行代码审查。
  - 持续集成：与 CI/CD 工具集成，自动化测试和部署。
  - ➢生态系统
  - GitHub：基于 Git 的代码托管平台。
  - GitLab：提供代码托管和 DevOps 功能的平台。
  - Bitbucket：支持 Git 和 Mercurial 的代码托管平台。
  - Git GUI 工具：如 Sourcetree、GitKraken 等，提供图形化界面。

项目需求说明

- 功能要求
  - 基本功能
    - git init：初始化一个新的Git仓库。
    - git add：将文件添加到暂存区。
    - git rm：从暂存区或工作区删除文件。
    - git commit：提交暂存区的更改，并生成一个新的提交记录。
  - 分支管理
    - git branch：创建、列出或删除分支。
    - git checkout：切换分支或回复工作区文件。

项目需求说明

- 功能要求
  - 合并功能
    - git merge：合并两个分支。
  - 进阶功能（可选）
    - git fetch：从远程仓库获取更新。
    - git pull：从远程仓库拉取更新并合并。
    - git push：将本地提交推送到远程仓库。
  - 补充信息
    - OJ平台只测试基本功能、分支管理以及合并功能。进阶功能可以写在文档中。线下演示时，可演示进阶功能实现效果。

项目需求说明

- 项目结构
  - 代码仓库
    - 每个小组在GitHub上创建一个仓库，命名为队名-git（如果仓库名已存在则加数字区分）。
  - 代码提交
    - 定期提交代码到GitHub仓库，确保代码的可追溯性。
  - 文档
    - 在仓库中提供README文件，详细说明项目的设计思路、功能实现、使用方法等。

评分标准

- OJ平台评分
  - 占总成绩的50%，基于公开测试用例和隐藏测试用例的通过情况。
  - 小队的代码提交后，会在OJ后台编译，我们将编译后的可执行文件拷贝到测试用例执行目录。假定编译后可执行文件为git，则会用编译后的git来执行Git系统的基本功能、分支管理与合并或进阶功能。
- 线下演示与问答
  - 占总成绩的30%，小组需要展示项目的功能，并回答老师的提问。
- 代码质量与文档
  - 占总成绩的20%，评估代码的可读性、模块化设计、注释、文档完整性等。

评分标准

- 测试用例设计
  - ➢公开测试用例
    - ◘基础测试用例1： git init
      - ◆描述：初始化一个新的Git仓库。
      - ◆步骤：
        1. 在空目录下执行git init。
        2. 检查是否生成.git目录，并且目录结构正确（如objects、refs等子目录）。
      - ◆预期输出：
        - ✓.git 目录存在，且结构正确。

评分标准

- 测试用例设计
  - ➢公开测试用例
    - ◘基础测试用例2： git add和git commit
      - ◆描述：添加文件并提交。
      - ◆步骤：
        1. 创建一个文件test.txt，内容为”Hello, Rust!”。
        2. 执行 git add test.txt。
        3. 执行 git commit –m “Initial commit”。
        4. 检查是否生成了一个新的提交记录，并且文件内容被正确存储。
      - ◆预期输出：
        - ✓提交记录生成，文件内容被正确存储。

评分标准

- 测试用例设计
  - ➢公开测试用例
    - ◘基础测试用例3： git branch和git checkout
      - ◆描述：创建分支并切换。
      - ◆步骤：
        1. 执行git branch test。
        2. git checkout test。
        3. 检查当前分支是否为test。
      - ◆预期输出：
        - ✓当前分支为test。

评分标准

- 测试用例设计
  - ➢公开测试用例
    - ◘基础测试用例4： git merge
      - ◆描述：合并两个分支。
      - ◆步骤：
        1. 创建一个main分支并切换到main分支。
        2. 在main分支上创建一个文件main.rs，内容如图1所示。
        3. 切换到test分支，创建一个文件test.txt，内容为`测试分支合并`。
        4. 切换到main分支，执行 git merge test。
        5. 检查main分支是否包含test.txt文件，调用rustc编译main.rs并执行。
      - ◆预期输出：
        - ✓main分支包含test.tx文件，执行`./main`得到输出`测试分支合并\n` 。

```rust
use std::fs::File;
use std::io::{self, Read};

fn main() -> io::Result<()> {
    // 打开当前目录下的 test.txt 文件
    let mut file = File::open("test.txt")?;

    // 创建一个字符串来存储文件内容
    let mut contents = String::new();

    // 读取文件内容到字符串
    file.read_to_string(&mut contents)?;

    // 打印文件内容
    println!("{}", contents);

    Ok(())
}
```

图1. 代码示例

评分标准

- 测试用例设计
  - ➢公开测试用例
    - ◘高级测试用例1： 冲突合并。
      - ◆描述：测试分支合并时的冲突处理。
      - ◆步骤：
        1. 在main分支上修改test.txt文件，内容为`main分支修改内容`。
        2. 切换到test分支，修改test.txt文件，内容为`test分支修改内容`。
        3. 切换回main分支，执行git merge test。
        4. 检查是否能够检测到冲突。
      - ◆预期输出：
        - ✓正确检测到冲突。

评分标准

- 测试用例设计
  - ➢公开测试用例
    - ◘高级测试用例2： 大文件处理。
      - ◆描述：测试大文件的添加和提交。
      - ◆步骤：
        1. 切换到main分支，创建一个大小为10MB的文件large_file.bin。
        2. 执行git add large_file.bin。
        3. 执行git commit -m "Add large file"。
        4. 检查文件是否被正确存储。
      - ◆预期输出：
        - ✓大文件被正确存储。

评分标准

- 测试用例设计
  - ➢公开测试用例
    - ◘高级测试用例3： 分支删除。
      - ◆描述：测试分支删除功能。
      - ◆步骤：
        1. 创建并切换到temp分支。
        2. 执行git branch -d temp。
        3. 检查temp分支是否被删除。
      - ◆预期输出：
        - ✓temp分支被删除。

评分标准

- 测试用例设计
  - ➢公开测试用例
    - ◘进阶测试用例1：远程操作（可选）。
      - ◆描述：测试远程仓库的拉取和推送。
      - ◆步骤：
        1. 在远程仓库中创建一个分支remote_branch。
        2. 执行git fetch，检查是否能够获取远程分支。
        3. 执行git pull origin remote_branch，检查是否能够合并远程分支。
        4. 执行git push origin main，检查是否能够推送本地提交。
      - ◆预期输出：
        - ✓远程操作成功执行。
      - ◆补充信息
        - ✓远程仓库可以用现有的代码仓库，也可以本地部署

《Rust程序设计语言》课程项目

基于Rust的Git系统

授课教师：冯洋

项目助教：燕言言

DG21320008@smail.nju.edu.cn

南京大学计算机学院