

计算机网络 - Tutorial1

授课助教：宋聿辰

songyuchen@stu.pku.edu.cn

(内容修改自 郭俊毅 2022秋 计算机网络 Tutorial1)

内容

- make和Makefile简介
- Cmake
- Docker
- Git & Github
- Github Classroom

内容

➤ make和Makefile简介

➤ Cmake

➤ Docker

➤ Git & Github

➤ Github Classroom

make & Makefile 简介

- **make** is a utility for organizing the files in a multi-file project
- Its purpose is to ensure that the changes to a source file will be propagated to the parts of project which depend on that file
 - ensures the minimum recompilation when something is changed
- make must have a list, called **Makefile**, containing
 - the files to be remade (**targets**)
 - the files upon which they depend (**dependencies**)
 - the commands needed to build each target

Motivation of make & Makefile

- **make** deals with multi-file project
- But why do we need multiple source files and objects?
 - small source file → a single program
 - “not-so-small” project:
 - not a good habit to have a source file with many lines of codes
 - better to have multiple source files
 - makes your program more structured and easier to improve your code
 - allows you to reuse object files in different projects
 - allows multiple programmers to work simultaneously

Motivation of make & Makefile - Example

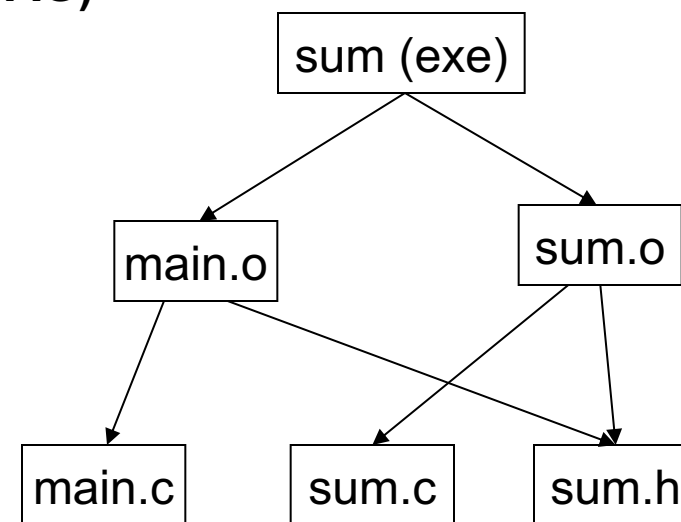
- Suppose you have written a file called `math.c`
 - it contains the code for your math functions
- You may want to compile the file to form an object file called `math.o`, using
 - **`gcc -c -o math.o math.c`**
- Suppose now you want to write a program that uses some of this math functions, and the main function of the program is in `main.c`
- You may compile `main.c` into an object file and link it with `math.o`
 - **`gcc -c -o main.o main.c`**
 - **`gcc -o myprogram main.o math.o`**

Motivation of make & Makefile - Example

- Problems with multiple source files
 - Multiple files are harder to manage
 - a simple change may require a long recompilation
- The objective of **make** is to
 - structure the source files using dependence relationships
 - ensures the **minimum recompilations** for any change in source files

Program structure

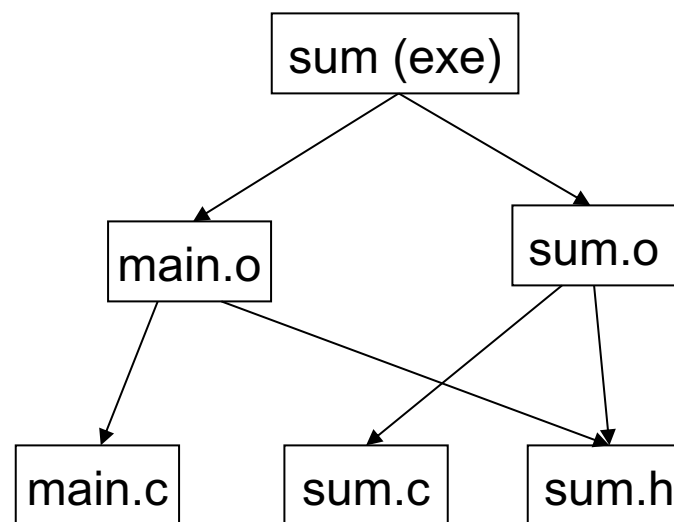
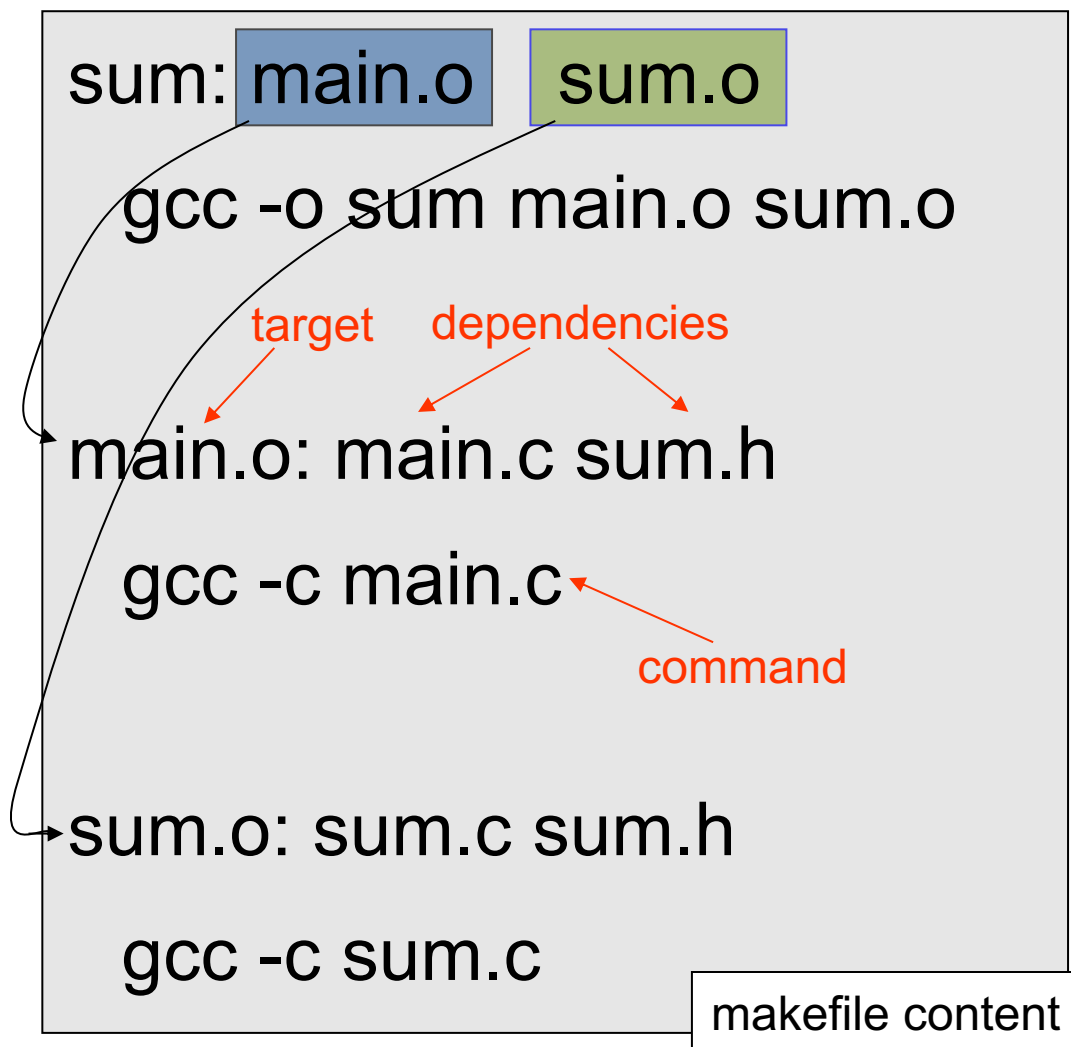
- **make** needs to know the structure of the program so as to organize the files in a multi-file project
- How to describe the structure?
 - we use dependence relationships
 - this relationship can be expressed by a **directed acyclic graph(DAG)**
- Example :
 - Program contains 3 source files
 - `main.c`, `sum.c`, `sum.h`
 - `sum.h` included in both `.c` files
 - Executable file `sum`



The make utility

- **make** is a utility in Unix which automatically maintains a program
- When you enter **make** command
 - **make** reads the file named **Makefile** or **makefile** in the current folder
 - it analyses the project structure and compiles the source files
- A **Makefile** is a file containing :
 - **Targets**: the files to be made
 - **Dependencies**: the program structure
 - **Commands**: Instructions for making the targets

A simple example of makefile



内容

➤ make和Makefile简介

➤ Cmake

➤ Docker

➤ Git & Github

➤ Github Classroom

CMake



- CMake是一个跨平台的、开源的构建工具
 - CMake是Makefile以及其他工具的上层工具，它们的目的正是为了可移植，并简化自己动手写更底层配置的巨大工作量

- 我们只需要如下两步:
 1. Write the CMakeLists.txt file
 2. Run cmake command

- CMake 目前是 C++ 的 de facto build system
 - 微软、Google 以及 Facebook 这三家公司都有自己的 C++ 构建系统，他们开源的项目仍支持使用 CMake 构建；并且 CMake 是除了官方构建系统之外的推荐构建系统

CMake的优势 & 一个Example

- 1. 和Makefile不同, Makefile是以文件为基础来维护要编译出来的目标的, 而CMake可以自定义一些更抽象的虚拟的目标来进行编译
- 2. CMake可以实现更复杂的功能, 包括但不限于: 跨平台的文件拷贝、根据一些选项生成对应的Makefile
- 接下来我们会展示几个例子, 分别展示CMake的如下几个功能:
 - 1. 如何编译一个可执行程序
 - 2. 如何编译一个静态库
 - 3. 如何将静态库链接到可执行程序中

1. 如何编译一个可执行程序

2. 如何编译一个静态库

3. 如何将静态库链接到可执行程序

更多CMake的常见命令

1. Specify the C++ Standard

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)

# set the project name and version
project(Tutorial)

# specify the C++ standard
set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED True)
```

如果CMAKE_CXX_STANDARD_REQUIRED设置为True，则必须使用CMAKE_CXX_STANDARD指定的版本

如果CMAKE_CXX_STANDARD_REQUIRED设置为OFF，则CMAKE_CXX_STANDARD指定版本的为首选版本，如果没有会使用上一版本。

更多Cmake的常见命令

2. 添加头文件搜索路径

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)
```

```
# set the project name and version
```

```
project(Tutorial)
```

```
# specify the C++ standard
```

```
include_directories(lib/ includes/)
```

如果将 lib 和 includes 目录都添加到到搜索路径的话，在 include 的时候就不需要使用相对路径了

More

- 一个非常推荐的cmake教程
 - <https://github.com/ttroy50/cmake-examples>

Ninja

- 细心的同学可能注意到了，我们在 Lab0 中让大家运行的指令是 `cmake .. -G "Ninja"` 而不是 `cmake .. -G "Unix Makefile"`。
- Ninja 是一个与 make 类似的编译管理工具，同样也可以作为 cmake 的后端使用。
- 理论上，使用 make 和 ninja 在编译结果上不会有区别
- （什么年代了，还在手写传统 makefile

内容

➤ make和Makefile简介

➤ Cmake

➤ Docker

➤ Git & Github

➤ Github Classroom

Docker 简介

- Docker 是一个基于容器的平台，能很方便地进行开发、打包应用程序。
- 在课程中，我们并不要求同学们掌握 docker 的使用方法，这里仅简单演示 docker 的常见命令。
 - 关于构建镜像和启动容器相关的配置文件，我们已经在 Lab0 中通过 Dockerfile 和 docker-compose.yml 下发给大家了。具体的操作步骤可以参考 Lab0 中的环境配置部分。
- docker ps, docker exec, docker run...

内容

➤ make和Makefile简介

➤ Cmake

➤ Docker

➤ Git & Github

➤ Github Classroom



Git & Github



- Git 是一个开源的分布式版本控制系统，常用于便捷高效地处理任何或大或小的项目。
- Git 是 Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。
- 我们强烈推荐你使用 git 管理你的项目。(因为我们本次使用了Github Classroom作为评测系统,所以我们也希望你至少掌握了git的基本使用方法以进行Lab的提交)
- GitHub是一个在线软件源代码托管服务平台，使用Git作为版本控制软件

教学内容

- 考虑到课程时间有限, 这里我们只针对Git的两方面内容进行教学:
- 1. 最基本的版本管理的指令
- 2. Git的分支和合并分支
- Ex. 一些常用的Git指令

基本的版本管理

➤ 首先我们在Github上创建一个空仓库

JeremyGuo ▾

Recent Repositories

New

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Repository template

Start your repository with a template repository's contents.

No template ▾

Owner *

Repository name *

JeremyGuo ▾ NetworkTest ✓

Great repository names are short and memorable. Need inspiration? How about [urban-robot](#)?

Description (optional)

☒ Public

Anyone on the internet can see this repository. You choose who can commit.

☐ Private

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ Add a README file

This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: None ▾

Choose a license

A license tells others what they can and can't do with your code. [Learn more.](#)

License: None ▾

① You are creating a public repository in your personal account.

Create repository

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH git@github.com:JeremyGuo/NetworkTest.git

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# NetworkTest" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:JeremyGuo/NetworkTest.git
git push -u origin main
```


...or push an existing repository from the command line

```
git remote add origin git@github.com:JeremyGuo/NetworkTest.git
git branch -M main
git push -u origin main
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

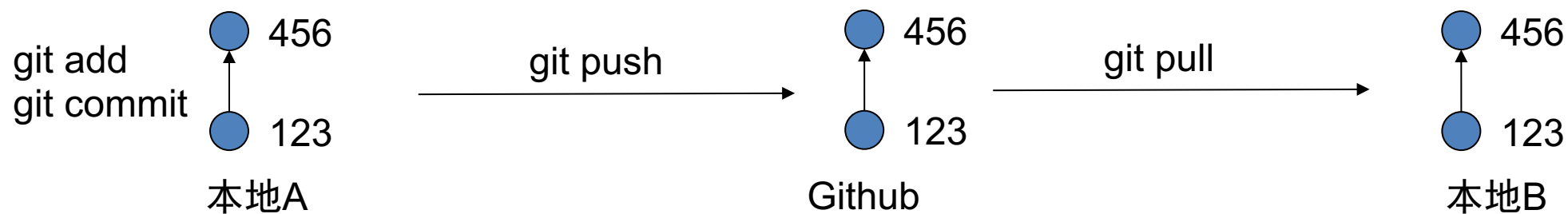
 北京大学
PEKING UNIVERSITY

《计算机网络》

26

基本的版本管理

- 此时我们已经成功创建了一个代码仓库
- 先来讲一讲push和pull, 这两个指令分别表示, 将本地的代码推送到仓库和将代码仓库的修改拉回到本地

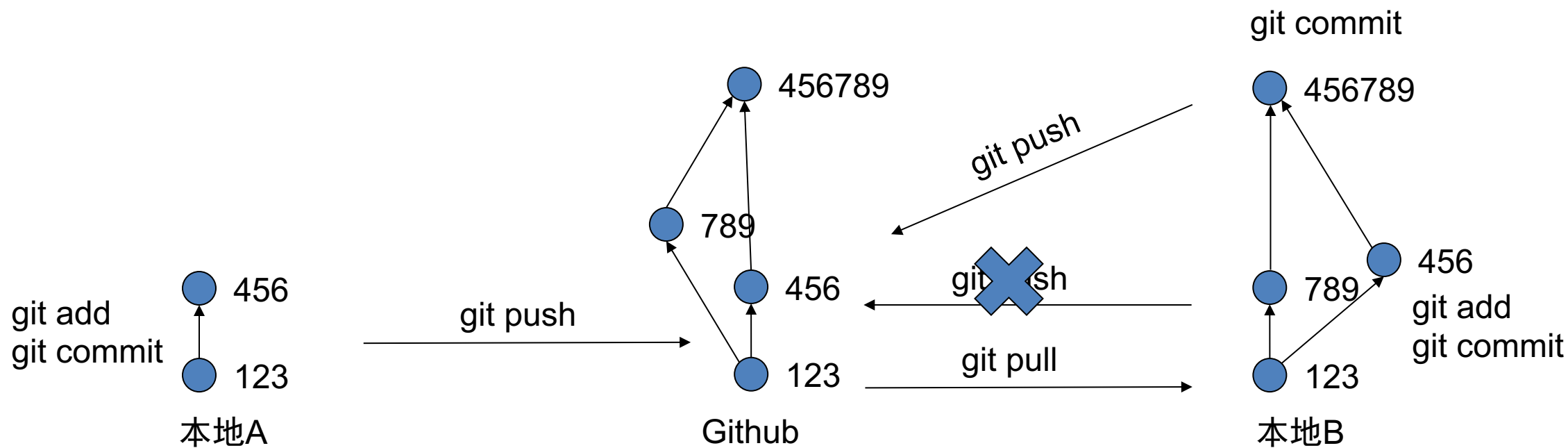


基本的版本管理

➤ 这里实际展示一下之前的例子

基本的版本管理

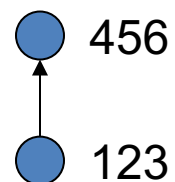
➤ 但是实际情况可能更加复杂



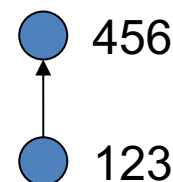
Git的分支和合并分支

➤ 注意到我们第一次打开Github仓库的时候是这个样子的

➤ 回到之前的例子



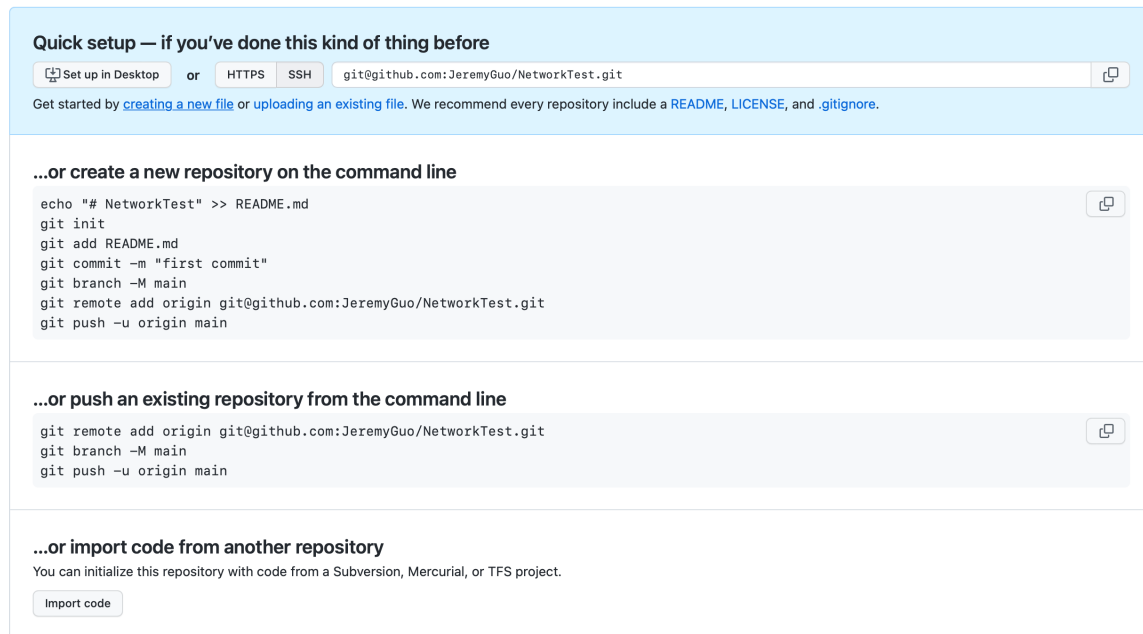
本地A(main)



Github(main)

➤ 每一个这样的树都有一个名字, 我们称这不同的名字叫做分支

➤ Github上看不到任何东西意味着我们没有任何一个分支



Git的分支和合并分支

- Github如果在创建仓库的时候选择创建README,那么默认会创建一个"main"分支
- 但是我们也可以手动管理分支

...or create a new repository on the command line

```
echo "# NetworkTest" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:JeremyGuo/NetworkTest.git
git push -u origin main
```

随便写点什么, 空文件夹无法Commit

告诉Git 该文件夹是一个仓库

告诉Git 下一次Commit应当包含这个文件

Commit

将当前分支重命名为main

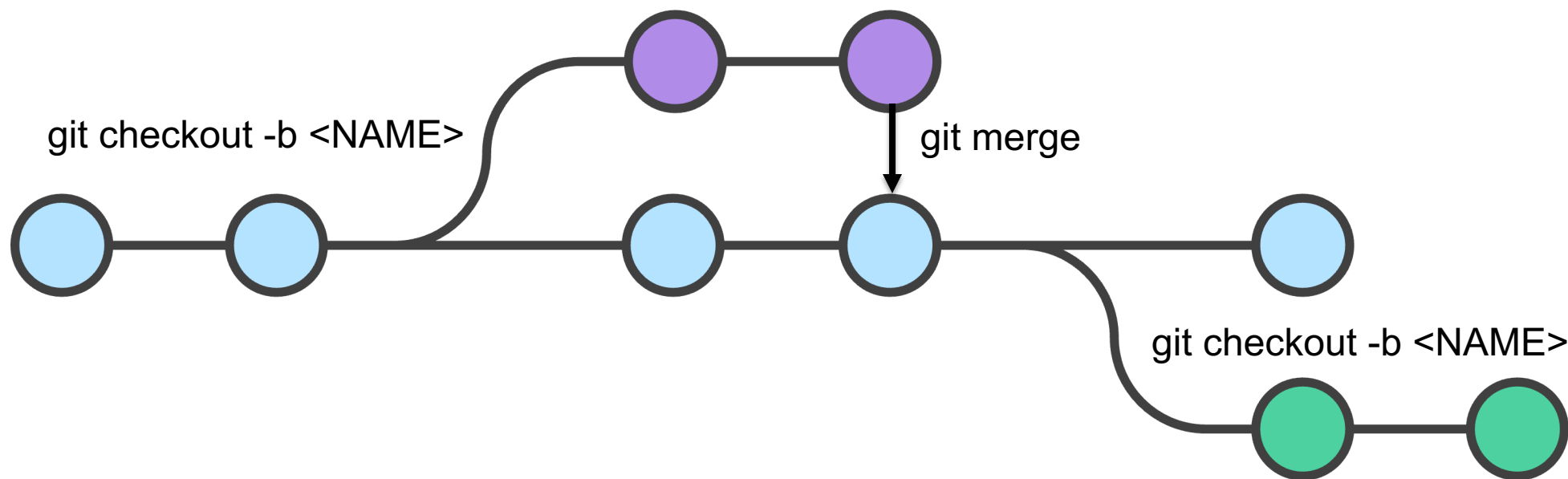
设置origin的地址

将本地main分支内容push到origin

-u 表示下次执行只需要git push

Git的分支和合并分支

- 在实际使用中, 我们最频繁使用的是 `git checkout` 和 `git branch` 指令
- 实际上创建一个分支可以从某一个当前的代码分支的节点开始创建



- 以之前的代码为例子, 我们实际演示一下这两个指令

Ex. 其他常用指令

- git log: 查看commit历史
- git status: 查看当前工作区状态
- git reset --soft HEAD^: 回退到上个commit
- git reset --hard HEAD^: 回退到上个commit, 并撤销当前工作区在上个版本上做的任何修改

内容

- make和Makefile简介
- Cmake
- Docker
- Git & Github
- Github Classroom

Github Classroom – Lab 0

- Lab 0不计入分数, 但是还是希望大家能简单做一做
- 完成该Lab大概需要5分钟时间, 具体的任务要求已经放到了课程实践主页上
- Example

Github Classroom – Lab 1

- Lab1是本学期的第一个Lab
 - 需要完成的任务是在TCP协议的基础上实现一个简单的应用层协议myFTP
 - 详见课程实践主页：<https://edu.n2sys.cn>
 - Example
 - 如何本地测试 & 如何提交
- DDL: 2023-10-31 23:30:00