



Introduction of SE-308 Software Design Projects

SE-308 Software Design Projects

(<http://my.ss.sysu.edu.cn/wiki/display/SDP>)

School of Software, Sun Yat-sen University

Outline

- **Syllabus**
- Introduction of Github collaboration workflow

课程概要

- **SE-308 《软件设计综合实验》**
- 综合使用各种软件设计技术，解决实际问题的能力
- 包括专题讲座和课程设计两部分
- 专题讲座指引学生了解最新IT技术与软件设计技术
- 课程设计中，学生组成项目小组，完成软件分析、设计与开发（**软件题目自选**）

- 课程学分（1学分）为申请制
- 学生组队完成软件设计综合实验后，按规定提交作品，参加答辩
- 课程考核组综合作品展示、答辩和作品评审情况，评定是否通过，并给出课程成绩
- **可使用其它项目申请**

专题讲座

- 6~8次专题讲座
- 具体题目、地点、时间、主讲教师等将在学院网站公布
- 学生**必须参加至少3次讲座**，讲座出勤情况将计入课程分数

课程设计

- 自行组织小组，在Github上，完成本课程设计

编号	任务	截止时间
1	完成组队并提交软件设计题目	4月30日
2	持续提交软件工程制品	
3	提交最终制品	6月下旬，待定
4	展示答辩	待定

1. 组队与题目

- 填写SoYa项目申请表，提交组队情况和软件题目，申请开通SoYa平台上的软件项目（学委收集后，提交给TA）
- <http://my.ss.sysu.edu.cn/wiki/pages/viewpage.action?pageId=33882183>

Project & Group

1 Added by 王霞, last edited by 王霞 on Apr 11, 2012 (view change)

⚠ 请于**5月20日**之前提交《**SoYa项目申请表**》，说明分组名单和软件项目。

下载模板：[SoYa申请表.doc](#), 文件名命名方式：**项目名.doc**

提交申请表

Name	Size	Creator	Creation Date	Comment
SoYa申请表.doc	101 kB	王霞	Apr 11, 2012 11:46	

Browse... Enter a comment for this file here Attach

Add Labels

- 软件题目不限，使用技术不限
- 每组4~6人
- 每人只能加入一个项目组

2. 项目开发、提交更新制品

- 使用Github，持续提交制品



- 项目小组成员必须自己提交工作，课程考核组根据Github中的Issue分派和Git提交记录，确定工作的归属

3. 提交最终制品

- 提交最后软件制品到Github，并打上**final**的**tag**。最后制品包括：
 - 安装包与《安装部署说明》
 - 《用户手册》或《使用说明》
 - 所有源代码
 - 《软件需求规格说明书》（**SRS**）
 - 《软件设计文档》（**SDS**）
 - 《小组分工与贡献率说明》

4. 展示答辩

- 现场安装、展示、说明软件
- 现场从**Github clone**项目安装包，执行安装、配置
- 现场运行、展示软件。课程考核组实操或指示演示者操作软件
- 基于《软件设计文档》，讲解说明软件设计思想（可使用幻灯片）
- 答辩，回答课程考核组问题

最终制品要求

- 安装包与《安装部署说明》
 - 清晰说明安装的环境、步骤
 - 安装成功的测试方法
 - 常见问题解决方法
- 《用户手册》或《使用说明》
 - 若软件界面与交互设计合理，无需说明
 - 若软件自带帮助系统，则可不提供
- 所有源代码
 - 软件源代码和测试代码
 - 如果代码模块、层次较为复杂，应给出目录说明与索引。
- 《软件需求规格说明书》（SRS）
 - SRS繁简皆宜，项目组可灵活选择
 - 标准是，对于所有系统需求，项目组成员必须基于SRS，有完全相同的理解。

最终制品要求

- 《软件设计文档》（SD）**重点～！**
- 详细说明软件的技术选型理由、架构设计、模块划分
- 列出项目中使用的软件设计技术，包括但不限于：
Structure Programming、Object-Oriented Programming、Aspect-Oriented Programming、Service Oriented Architecture、Design Patterns中的种种技术
- 对于每种使用的技术，SD应当**给出具体设计在源代码中出现的位置**，指明对应模块和代码。

- 课程考核组将**重点考核SD**，对照代码，逐一落实SD罗列的架构设计、模块划分与软件设计技术，据此给出软件可扩展性分数和可维护性分数。

最终制品要求

- 《小组分工与贡献率说明》
- 至少包括《小组分工与贡献率》 和《制品与贡献率》 两个列表
- 课程分数评定时，考核组将对照**Git log**中的提交记录，进行检查，如与实际情况不符，将按照课程考核处罚扣分

最终制品

- 《小组分工与贡献率》和《制品与贡献率》

表 2 小组分工与贡献率（示例）

学号	姓名	分工	贡献率（合计 100%）
1234567	张三（组长）	项目管理、需求分析、Module1 设计与 Coding、答辩	27%
2345678	李四	需求分析、Module2 Module3 设计与 Coding	22%
3456789	赵五	架构设计、Module1~Module4 设计与 Coding	28%
4567890	王六	UI 设计、交互设计、Module4 设计、项目文档整理	23%

*小组成员根据实际分工和完成情况，自行商定贡献率。

表 3 制品与贡献率（示例）

	制品*	张三	李四	赵五	王六
源码	Module1	60%	-	40%	-
	Module2				
	Module3				
	Module4				
文档	SRS				
	SD				

原创性注意事项

- 源码和文档必须为项目小组成员原创
 - 软件使用开源或他人代码与文档的部分，必须在SD当中说明，并且不得列入贡献率表格。违反者视为抄袭。
- 可以采用其它项目作为本课程设计
 - 但所有源码和文档必须为本小组成员原创。采用其它课程项目时，必须在SD和《小组分工与贡献率说明》中明确指出。违反者亦视为抄袭
- 务必按照小组成员实际工作，向Github分别提交制品，以便课程考核组检查。**课程考核组将以Git log为准，确定工作归属，不接受其它任何解释。**

课程考核

个人课程分数 = 课程考勤分数 × 30% + Min(软件分数 × 小组人数 × 个人贡献率, 100) × 70%

课程考勤分数 = Min(讲座出勤次数 × 100 / 3, 100)

软件分数 = (完整性 + 正确性 + 可扩展性 + 可维护性) × 规模系数

个人贡献率：为《小组分工与贡献率说明》中给出的贡献率。

1. 完整性 (15 分)

- a) 文档齐备并内容完整 (5 分)
- b) 软件安装配置正确并顺利启动 (10 分)

2. 正确性 (15 分)

- a) 现场展示无 Bug (10 分)
- b) 课程考核组测试无 Bug (5 分)

课程考核

3. 可扩展性 (35 分)

- a) 技术选择合理 (5 分)
- b) 架构合理 (10 分)
- c) 模块化合理 (10 分)
- d) 模块内设计 (类、属性、方法, 或函数、数据结构) 合理 (10 分)

4. 可维护性 (35 分)

- a) 文档中关键模型 (用例图、Domain Model 或 ER 图、架构图) 存在并正确 (10 分)
- b) 代码可读性 (15 分)
- c) 测试用例 (测试代码、测试数据与必要的说明) 覆盖关键用例和关键代码 (10 分)

课程考核

5. 规模系数 (0~1):

- a) 一般情况下，规模系数为 1。
- b) 如果存在下列情况之一，课程考核组调查分析后，可认定软件规模过小。
 - i. 实现用例不足 5 个
 - ii. 代码源文件总个数不足 20 个
 - iii. 代码行总计不足 2000 行

软件规模过小时，由课程考核组综合考虑软件领域和具体技术，给出规模系数 (0~1)。

处罚与扣分

处罚与扣分

1. 抄袭。抄袭等同于作弊，小组全体分数零分，并按照学院规定处理。
2. 《小组分工与贡献率说明》不实。不论个人贡献率高于实际情况或低于实际情况，均视为与实际情况不符。

不符者分数 = Max (软件分数 × 小组人数 × 小组最低贡献率 - 10, 0)。

3. 展示答辩时，软件未能成功安装。
 - a) 软件完整性分数扣除 10 分，软件正确性分数为零分。课程考核小组视情况给出其余各项分数。

课程教学网站

- <http://my.ss.sysu.edu.cn/wiki/display/SDP/Home>

The screenshot shows the homepage of the SE-308 Software Design Projects wiki. At the top, there's a navigation bar with links for Dashboard, SE-308 Software Design Projects, Home, Browse, Wang Wei, and Search. Below the navigation is a header with a blue 'X' icon and the word 'Home'. A message indicates it was added by Wang Wei on April 11, 2012, with a link to view changes.

Schedule

Date	Event
13 April 2012	09:50 逻辑介绍 行政楼讲学厅
20 May 2012	All day 组队与软件设计题目提交截止
21 July 2012	All day 提交最迟题目截止
28 July 2012	All day 答题

Calendars

- System Design Projects
- Course
- Add a calendar

News

王伟 posted on Apr 11, 2012

组队与软件设计题目选定

To Project & Group, download and fill out the [SoYa Project Application Form](#). Submit team information and software projects to apply for software projects on the SoYa platform.
1. Software projects are not limited, and the technology used is not limited.
2. Each group has 4-6 people.
3. Each person can only join one project group.
Deadline: **May 20**.

[Edit](#)

Outline

- Syllabus
- **Introduction of Github collaboration workflow**

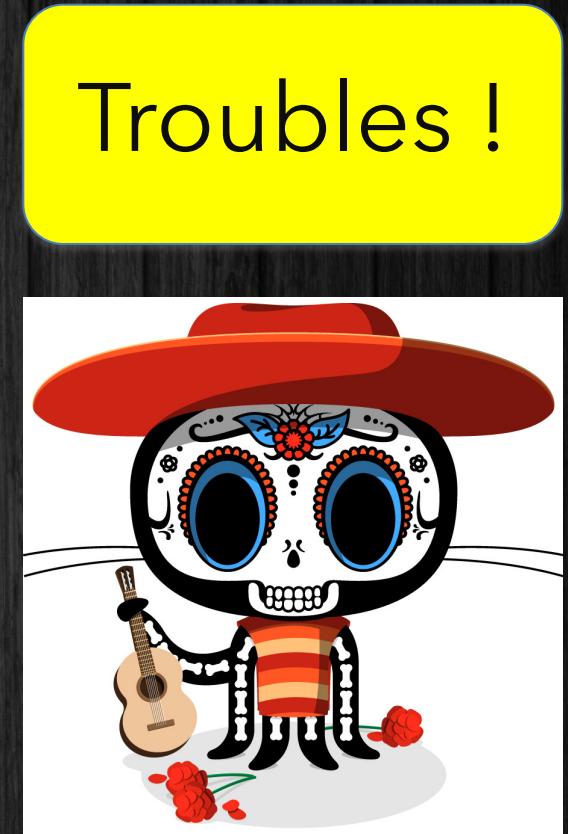
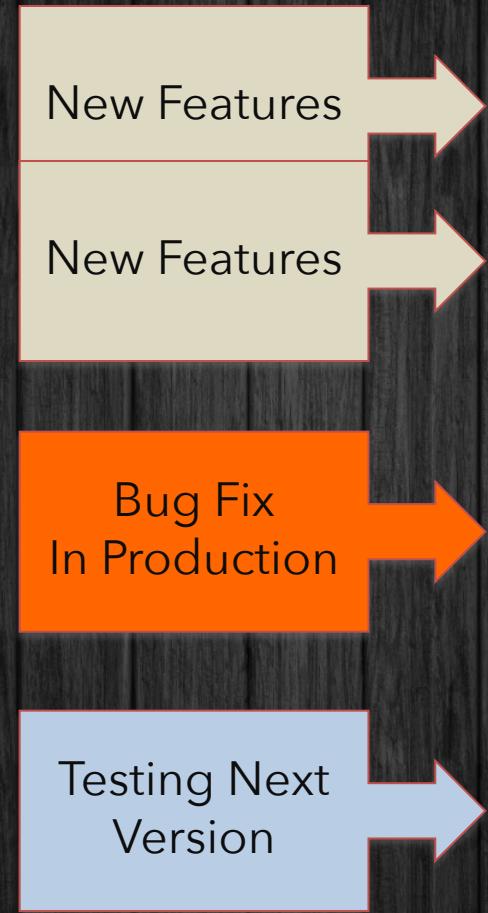
The origin of the problem



How code lives

lifecycle

Projects and code evolve with time



Solutions ?



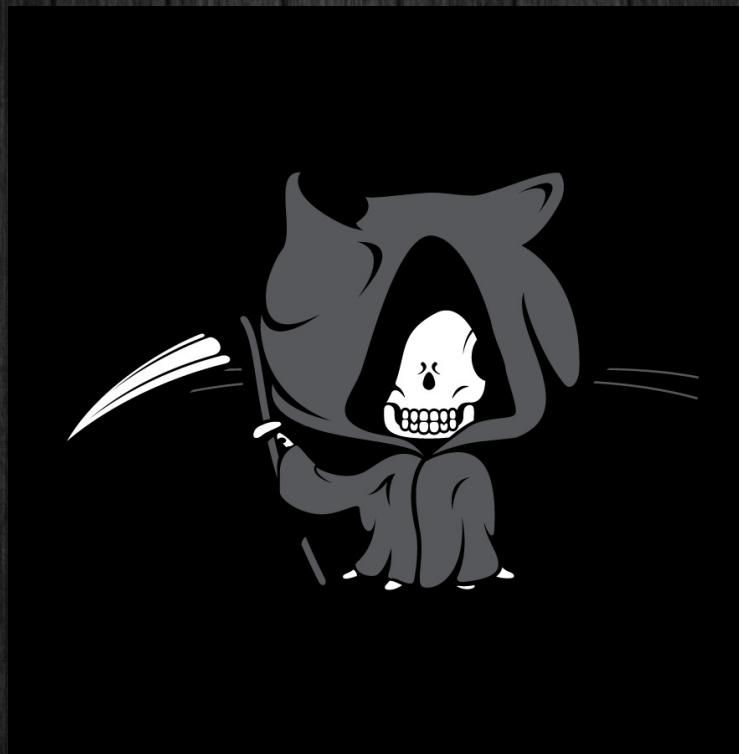
How people code

@rhw // Rui Carvalho

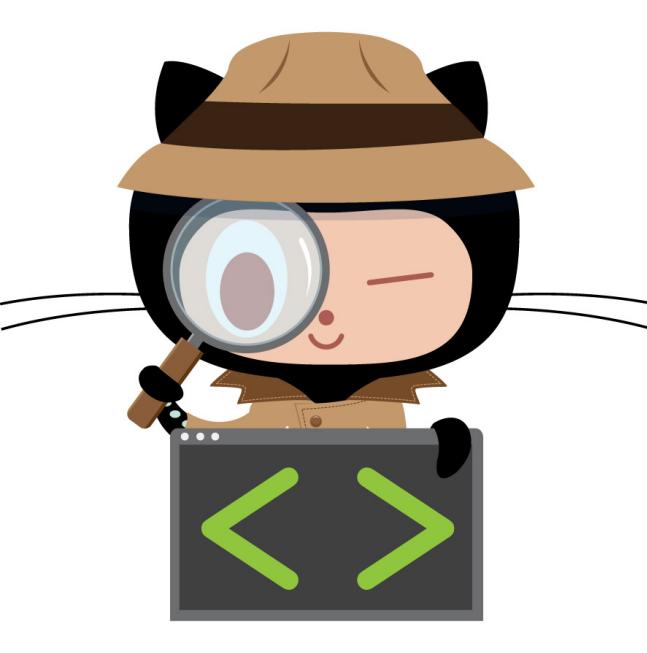
Classic VCS Habits

- Do a project “*Long running*” branch
- Continue that **ugly** coding
- Pray to be the **first** to merge
- Deploy **monthly** to avoid conflicts
- **Iterate** for X years (average 5?)
- **Big bang** the project and create a new VCS root

Sounds good ?



@rhwy // Rui Carvalho



Should I care ?

Source Code

Code Matters!

- Your **code** and your **VCS** are your only valid **documentation**
- Your code is **living** and your VCS is the open book of that **History**



Keep calm and code



Rules

@rhwy // Rui Carvalho

Objectives ?

- **Always** be able to **build** a **production** version
- Always be able to build a **stable** version
- **Easily** integrate new **features**
- **Avoid** long term “out of space” code (aka **project branches**)
- Produce **Value** on each **commit**



Simple model

git branch

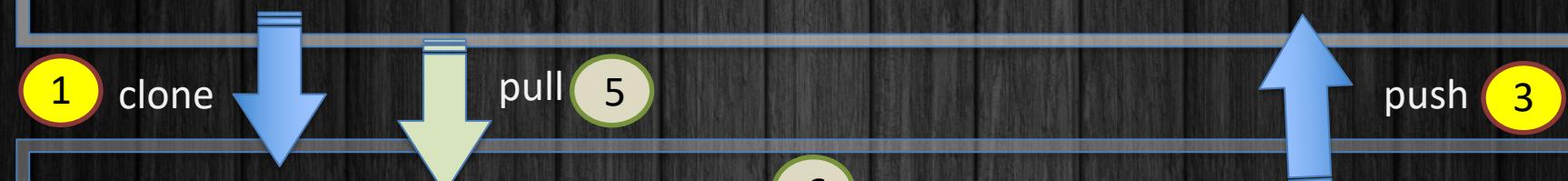
Small branches

- Very **small** branches
- Keep **central** repository **clean**
- Work locally then **push** your branch
- Someone else **review** & **merge**
- **Update** your master

Small branches

- **No** learning curve
- **Near** classic VCS patterns
 - But, better **isolation**
 - Easier, due to **no-cost** branching
- Work **remotely**
- No **side** effects

Distant



Local

```
//2 step branching  
$> git branch "create_login_box"  
$> git checkout create_login_box
```

```
//1 command branching  
$> git checkout -b create_login_box
```

Create Branch, then Checkout

```
$> //add files, do some code  
$> git add loginViewModel.cs  
$> git commit -m "created view model for login"  
$> //more core, finish  
$> git commit -m "created html login UI"  
  
//then push your branch with that commits  
$> git push origin create_login_box
```

Code, Add to index, Commit, push branch

```
$> //on the remote repo  
$> git checkout master  
$> git create_login_box  
  
$> //on your box, sync the master from remote  
$> git pull origin master
```

Someone merge,
then you can update your local master

```
$> //if you need to continue on the repo, sync  
it with the updated master  
$> git checkout create_login_box  
$> git rebase master  
  
$> //if not delete it  
$> git branch -d create_login_box
```

Then continue your work,
Or simply delete your branch

A better model



git fork

@rhwy // Rui Carvalho

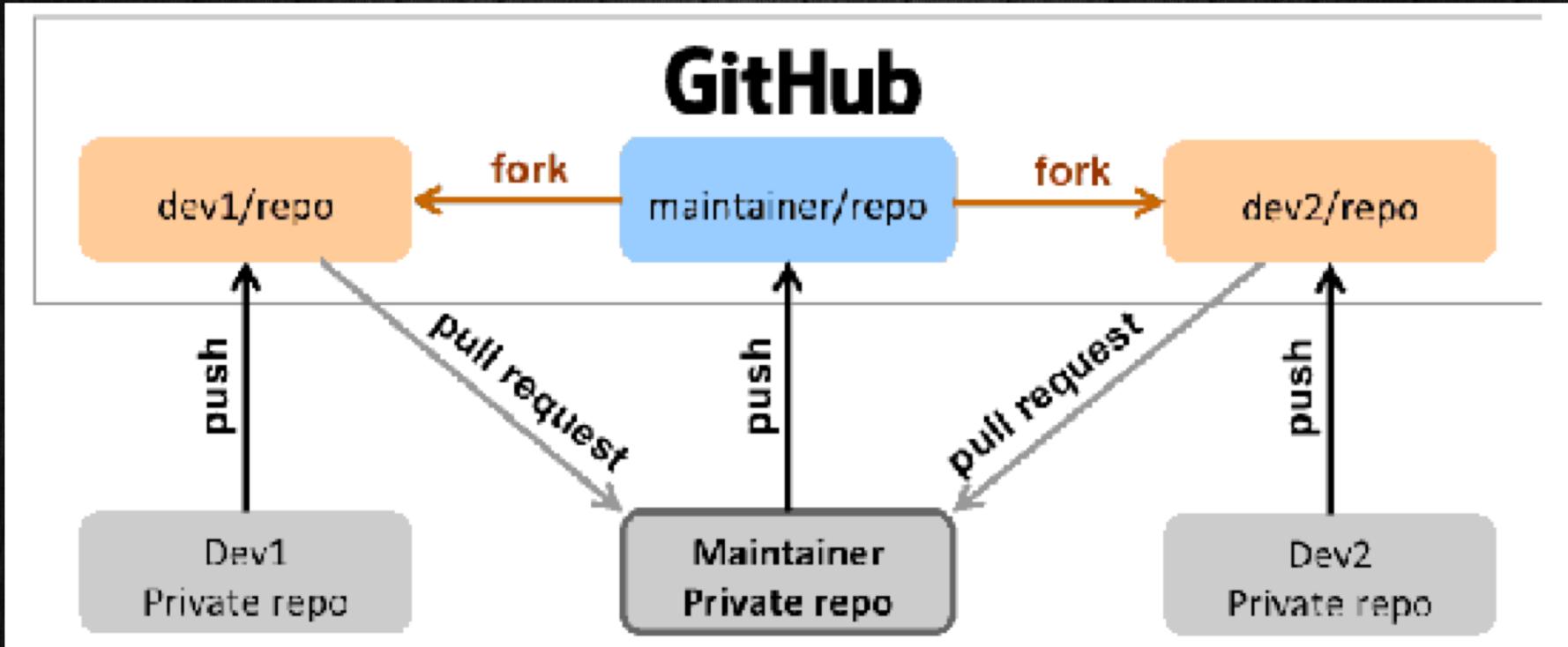
Fork what?

- **Protect** your central repository
- **Restrict** rights, accept **external** work
- **Your** branches, your **mess**
- Your own **builds**, easier **team** work
- **Not git**, but quite all online services support it
 - = **clone on server side**

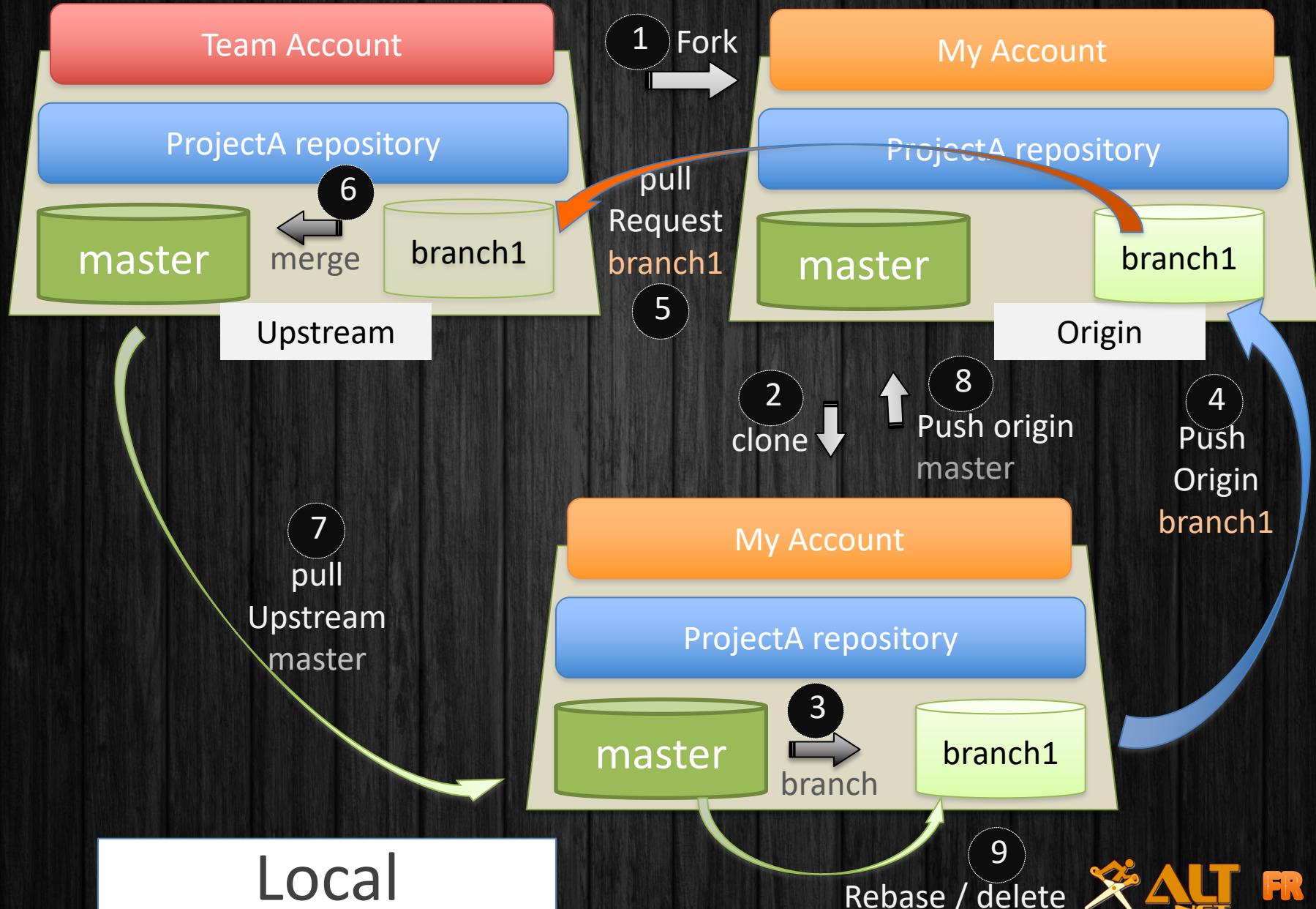
Other benefits

- Easier **remote** work
- YOU, sure to always have a **clean master**
- Easier **code review** across teams
- It opens a **discussion**

GitHub



Distant



Local

```
$> //remotely:  
$> //fork it/clone it on the server side from  
company/projectA to myaccount/projectA
```

```
$> //locally:  
$> git clone myrepo/projectA  
$> git checkout -b my_feature
```

1. Fork, clone & branch

```
$> git add myfile  
$> git commit -m "I added some value"  
  
$> git push origin my_feature
```

2. Work, add, commit, push origin

```
$> //remotely:  
$> //pull request : ask to push your branch  
my_feature from myaccount/projectA to  
company/projectA
```

```
$> //then merge on company/projectA  
account (probably someone else)
```

```
$> //locally:  
$> //update your master with the merged one  
$> git pull upstream master
```

3.Pull request, merge, re-sync

```
$> //locally  
$> //to be sure to always have a clean master  
available everywher, you need to update it  
$> git push origin master  
  
$> //continue to work  
$> git checkout my_feature  
$> //sync your branch from updated master  
$> git rebase master  
$> //if not delete it  
$> git branch -d create_login_box
```

Then, update your remote fork,
And continue your work,
Or simply delete your branch

Better code, more fun



@rhwy // Rui Carvalho

Thank you!