

# Welcome to RES 2019

RES, Lecture 00

---

Abraham Rubinstein  
Guillaume Serneels



HAUTE ÉCOLE  
D'INGÉNIERIE ET DE GESTION  
DU CANTON DE VAUD

[www.heig-vd.ch](http://www.heig-vd.ch)



<https://github.com/arubinst>

<https://github.com/gsern1>

# Agenda

---



HAUTE ÉCOLE  
D'INGÉNIERIE ET DE GESTION  
DU CANTON DE VAUD  
[www.heig-vd.ch](http://www.heig-vd.ch)

- **Background**
  - Who are we?
  - What do we do at the HEIG-VD and why?
- **Course objectives**
  - Network programming
  - Application-level protocols
  - Web infrastructures
- **Tools**
  - Crash course on Git & GitHub

# Background



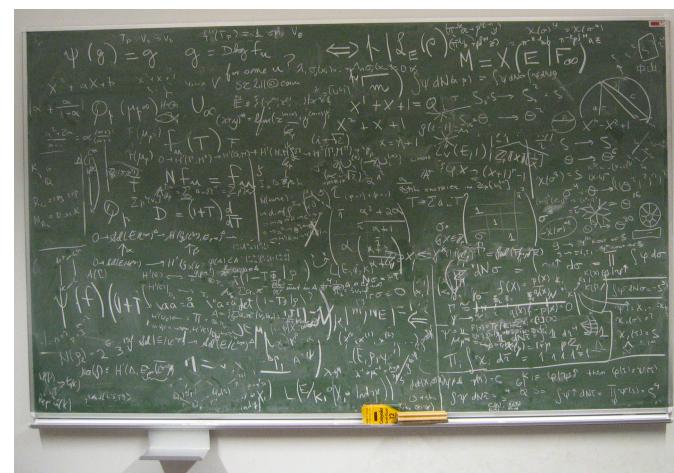
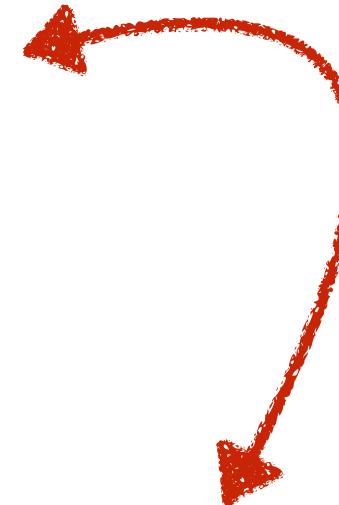
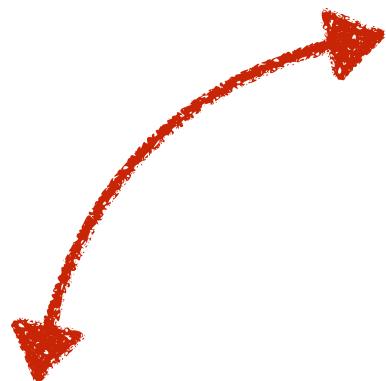
# Personal background



# Personal background



# Why do we do applied research?



# There are plenty of opportunities!



HAUTE ÉCOLE  
D'INGÉNIERIE ET DE GESTION  
DU CANTON DE VAUD  
[www.heig-vd.ch](http://www.heig-vd.ch)

## STarmac

**Une initiative de la HEIG-VD mettant à votre disposition les outils nécessaires pour devenir un startuper accompli.**



## STarmac

Depuis de nombreuses années, la HEIG-VD stimule et soutient l'innovation et l'esprit d'entrepreneuriat au travers de formations, de coaching, de prix et distinctions, mais aussi de liens avec les organismes de soutien à l'innovation de la région.

Aujourd'hui, ces différentes initiatives se rassemblent sous un même nom et en un même lieu : STarmac (le Tarmac pour le Startups).

Il s'agit d'un espace privilégié d'échange et d'émulation à disposition de celles et ceux, étudiants ingénieurs ou économistes et collaborateurs, qui veulent partager, mettre à l'épreuve ou renforcer leur désir d'innovation et leur passion d'entreprendre.



**Présentation**

**Organisation**

**Partenaires**

**Business Concept**

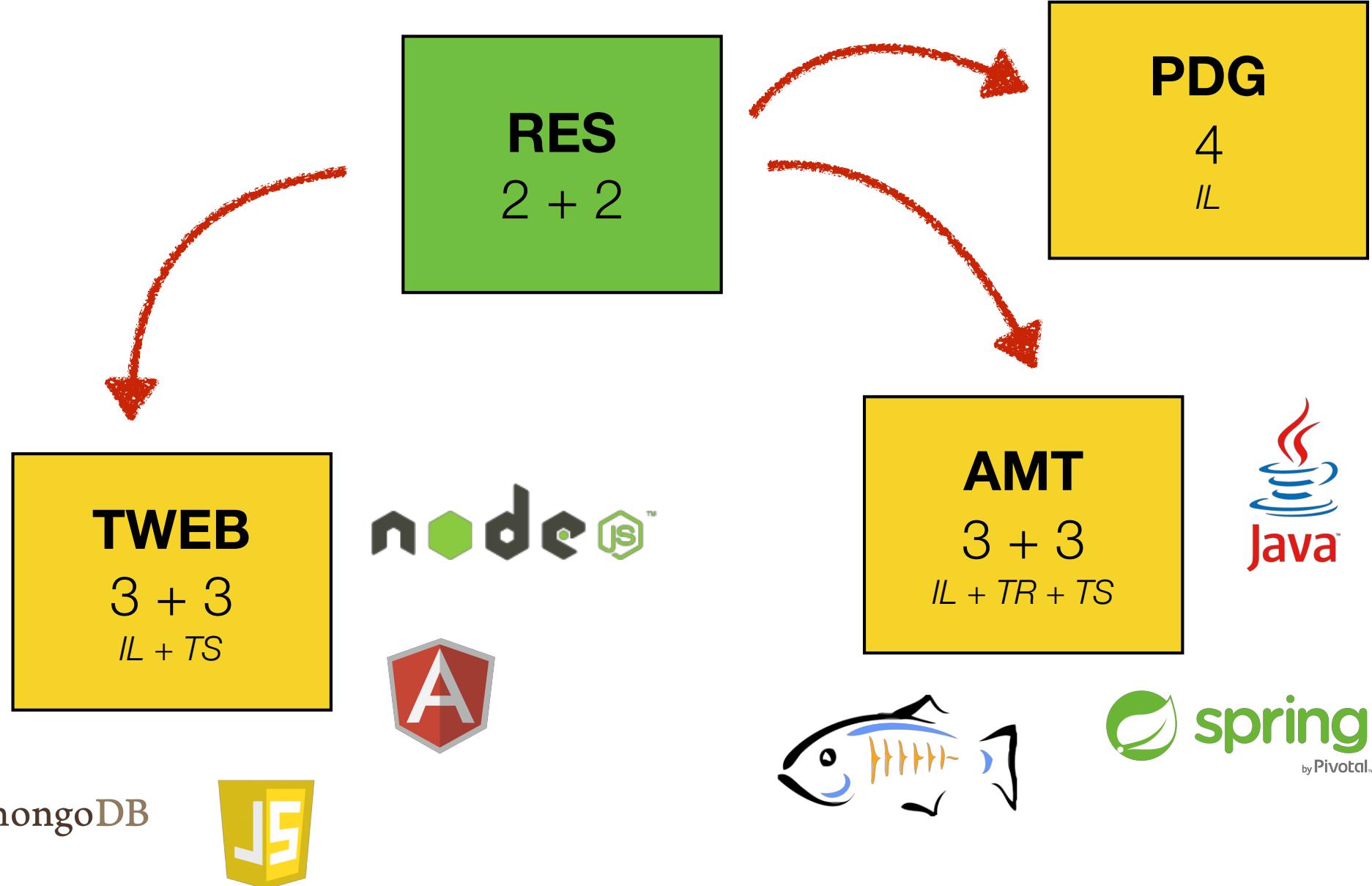
**Business Validation**

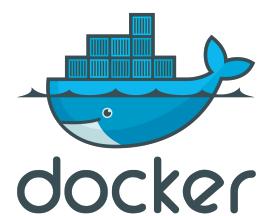
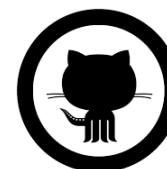
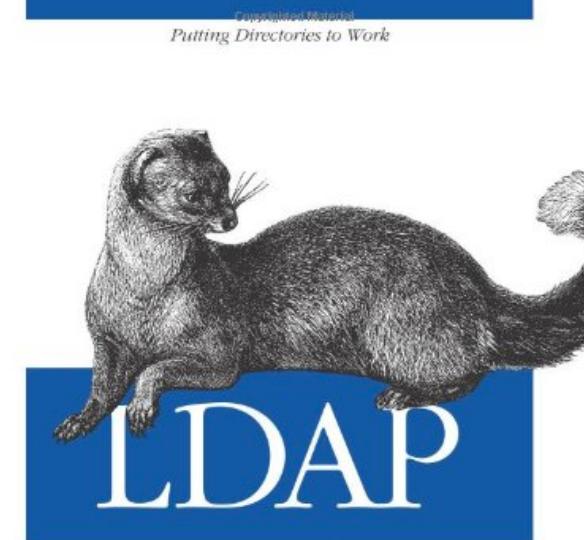
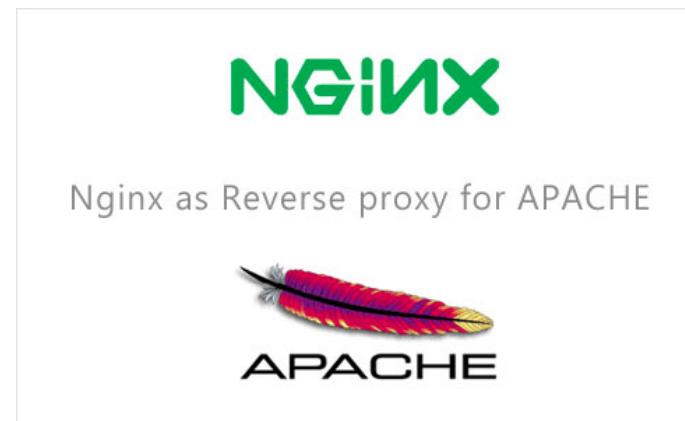
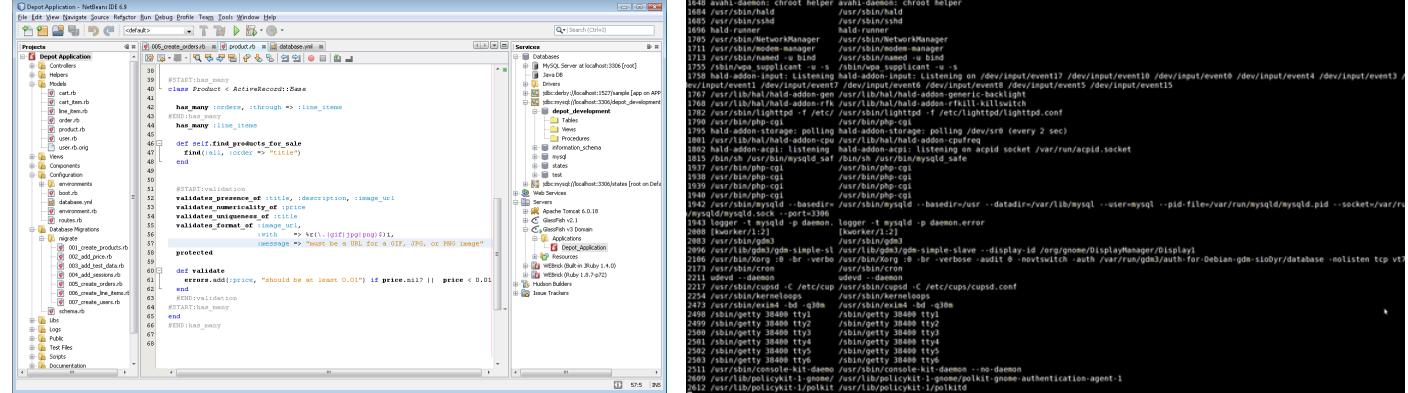
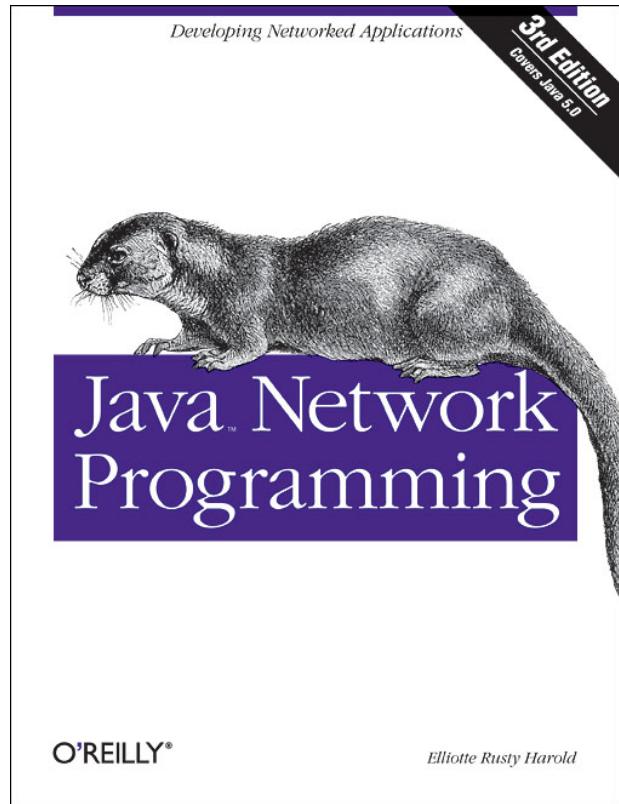
**InnoGrant**

**Contact**

# Course Objectives







# High-level planning

- **IO programming** in Java (starting with files, encodings, etc.)
- **Network programming** in Java (and Javascript)
  - How do we write TCP clients and servers?
  - How do we use UDP in our own programs?
- The **SMTP protocol**
- The **HTTP protocol**
  - Model, syntax, mechanisms
- Design and implementation of **web infrastructures**
  - Servers, proxies, reverse proxies, load balancers
  - Building a data center on your laptop with Docker
- *(Managing user identities with **LDAP**...)*

# GAPS: plan d'études



## Conditions pour la programmation automatique de cette unité selon le plan d'études :

L'étudiant-e doit avoir obtenu une note supérieure ou égale à la limite de compensation dans les unités : [TIB](#)

L'étudiant-e doit avoir suivi ou suivre en parallèle les unités : [POO1](#)

## Objectifs

*Ce champ est obligatoire*

- Programmation Réseau
  - Etre capable de concevoir une application client-serveur
  - Etre capable d'implémenter un client et un serveur en utilisant la Socket API dans différents langages
- Protocole HTTP
  - Connaître les concepts principaux du protocole
  - Etre capable de concevoir et réaliser une infrastructure HTTP avec un reverse proxy et plusieurs serveurs
  - Etre capable d'implémenter le protocole en utilisant la Socket API
- Protocole LDAP et annuaires Internet
  - Connaître le modèle LDAP et les éléments principaux du protocole
  - Etre capable d'installer et de configurer un serveur LDAP
  - Etre capable d'utiliser un client LDAP pour accéder à un serveur
  - Etre capable de transformer et d'importer des données dans un annuaire
- Protocoles de messagerie
  - Connaître les principaux protocoles relatifs à la messagerie électronique
  - Etre capable d'implémenter un client de messagerie simple
- Protocoles de transfert de fichiers et d'accès à distance
  - Connaître les protocoles de transfert de fichiers et d'accès à distance, ainsi que leurs principales utilisations (y compris tunneling/forwarding)
  - Etre capables d'utiliser des outils de synchronisation de fichiers à distance (e.g. rsync, ...)

# GAPS: plan d'études



HAUTE ÉCOLE  
D'INGÉNIERIE ET DE GESTION  
DU CANTON DE VAUD  
[www.heig-vd.ch](http://www.heig-vd.ch)

<b>Cours</b>	<b>32</b>
Concepts de programmation réseau et présentation de la Socket API dans différents langages	10
HTTP: étude du protocole et des éléments liés à l'infrastructure (e.g. reverse proxy)	10
LDAP: étude du modèle, du protocole et des éléments liés à l'infrastructure	6
Messagerie: études des protocoles principaux	4
Protocoles de transferts de fichiers et d'accès à distance: études des protocoles et outils (e.g. rsync)	2
<b>Laboratoire</b>	<b>32</b>
Développement d'une application client-serveur	10
HTTP: Développement d'un client et/ou d'un serveur simple	4
HTTP: Conception et implémentation d'une infrastructure avec un reverse proxy et plusieurs serveurs	6
LDAP: Mise en oeuvre d'un serveur, conception d'un schéma et import de données	6
Messagerie: implémentation d'un client simple	4
Mise en oeuvre des outils (e.g. rsync)	2

# Evaluation



HAUTE ÉCOLE  
D'INGÉNIERIE ET DE GESTION  
DU CANTON DE VAUD  
[www.heig-vd.ch](http://www.heig-vd.ch)

## Contrôle de connaissances

*Ce champ est facultatif*

**cours:** l'acquisition des matières de cet enseignement sera contrôlée au fur et à mesure par des tests et des travaux personnels tout au long de son déroulement. il y aura au moins 2 tests d'une durée totale de 2 périodes.

**laboratoire:** ils seront évalués sur la base des rapports de manipulation, à 3 reprises au minimum

## Note finale

Cours	25%
Laboratoire	25%
Examen	50%

- There will be 1 “standard” written test (~ mid semester)
- The second “written test” will be the result of multiple challenges:
  - On a regular basis, I will ask you to **complete short assignments** (quizz, small exercise, single development question, etc.). **Often, the assignment will need to be completed during the class or the lab session!**
  - Often, you will either get ‘1’ or ‘0’ point for completing the assignment or not. There might be assignments with more weight, hence more points. There will be easy wins and harder ones. I might do a quiz at the end of the lecture, and ask questions about what was presented on that day.
  - At the end of the semester, we will compute the grade of the 2nd “written test” based on these challenges.
  - Note on labs: sometimes, we will do a test after the handout to check that everyone in the group worked on assignment.
- **At the end of the semester, we will do a BIG lab (http infrastructure). It will have a heavy weight in the lab grade.**



Do not forget there is an **exam**  
Do not forget there are **lecture notes**

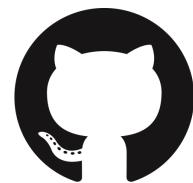


Deadlines are ridiculously strictly enforced  
We will not look at assignments late by 1'  
Do not blame the infra

# How not to get lost...



[https://github.com/arubinst/  
Teaching-HEIGVD-RES-2019](https://github.com/arubinst/Teaching-HEIGVD-RES-2019)



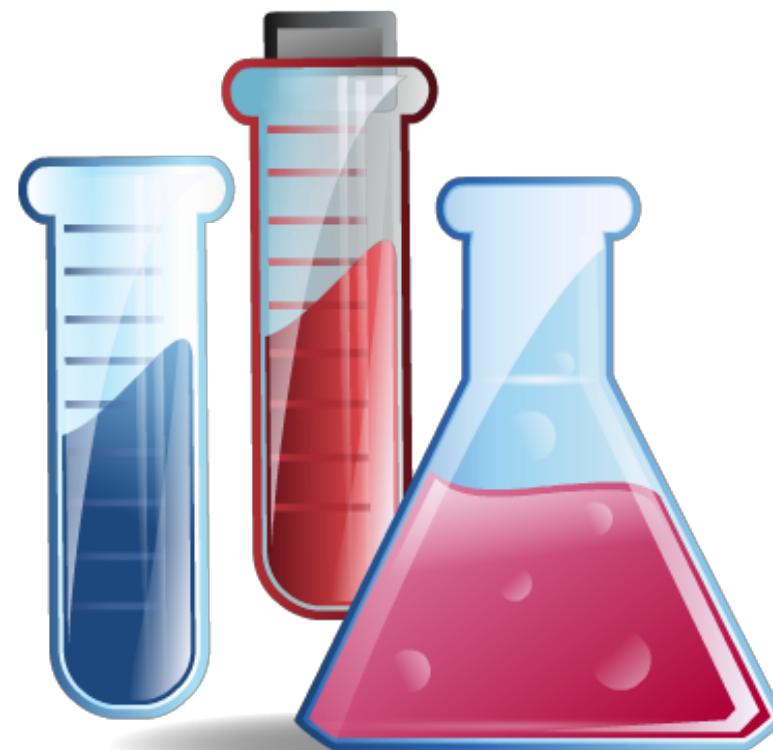
<https://goo.gl/forms/Ulmj9DWpV4hh8VqV2>



**10 minutes**

**Please always use the same Google and GitHub accounts**

# Lab Introduction



# Important points

---



HAUTE ÉCOLE  
D'INGÉNIERIE ET DE GESTION  
DU CANTON DE VAUD  
[www.heig-vd.ch](http://www.heig-vd.ch)

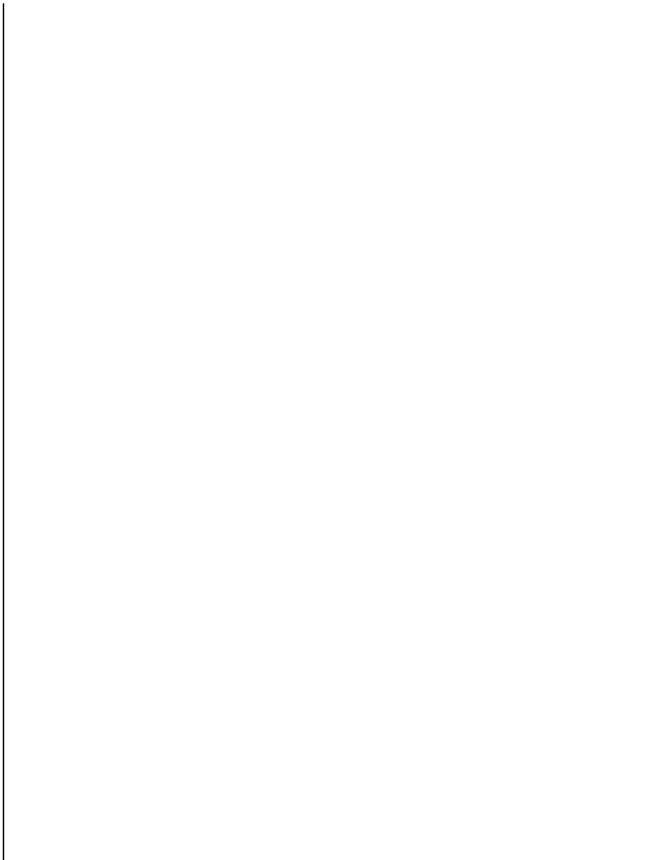
- **GitHub workflow** (start by forking, not cloning the source repo)
- What is **maven**?
- What is **project Lombok**?
- How do we use **unit tests** to specify behavior?
- **Where to write your code** (NOT in the test package!!!)
- What is an **application-level protocol** and how does not specify one?

# The Chill Protocol

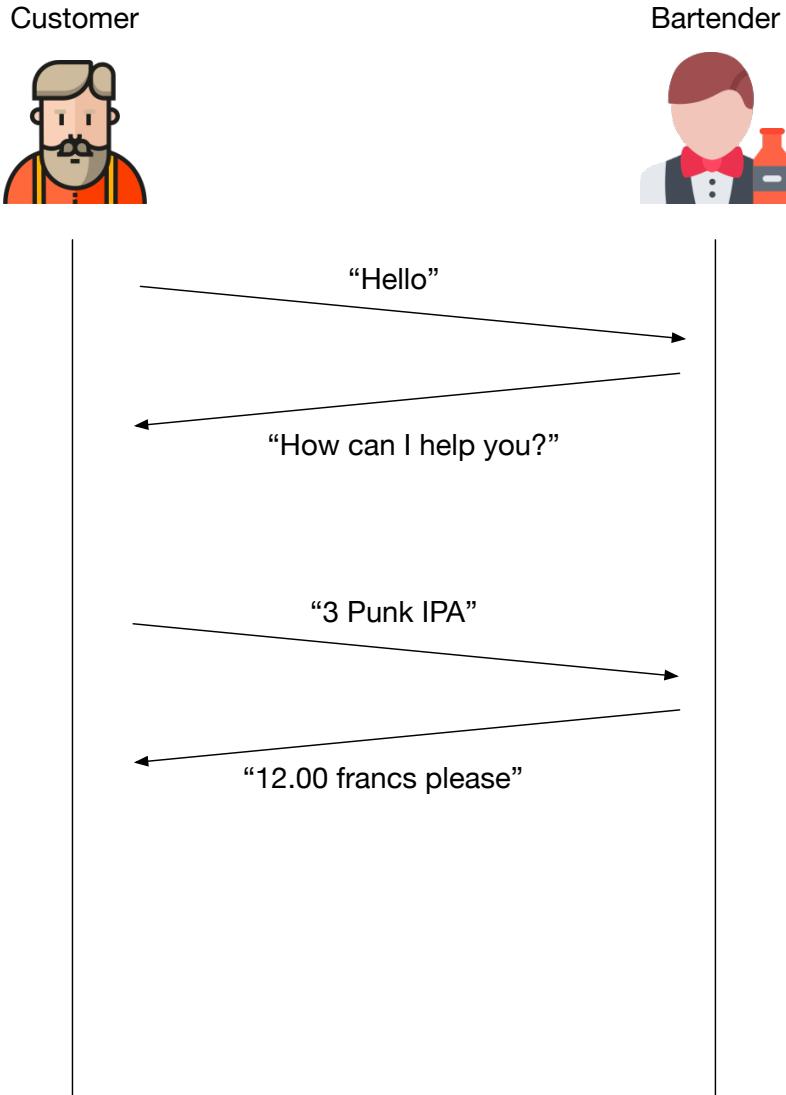
Customer



Bartender



# The Chill Protocol



## Entities

Server (providing a service)  
Client (requesting the service)

## Messages

Greeting  
Greeting response  
Order  
Order response

## Syntax

What is the structure of the messages?

## Sequencing

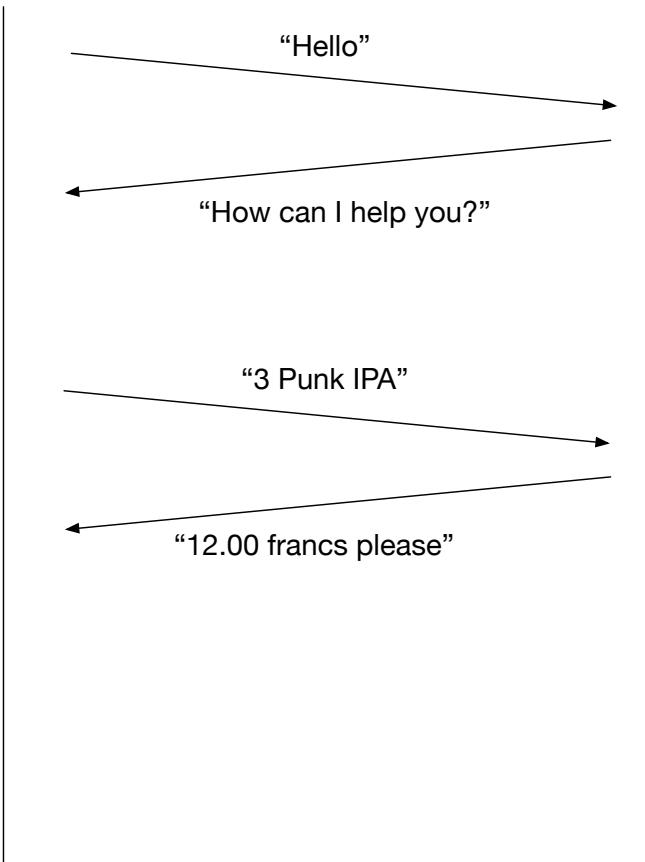
Order in which messages are exchanged  
How do entities react to messages?

# The Chill Protocol

Customer



Bartender



## Questions

- What happens if the customer does not say hello?
- What happens if the customer says hello twice?
- What happens if the customer order a beer that is not available?
- What happens if the customer is not 16 years old?
- What happens if the customer orders a burger and it takes 15' to prepare?
- What happens if the bar is crowded?

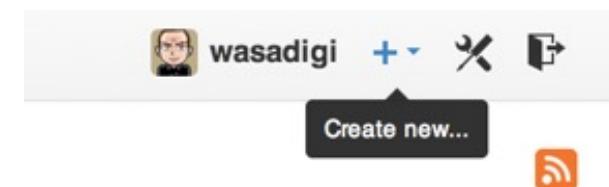
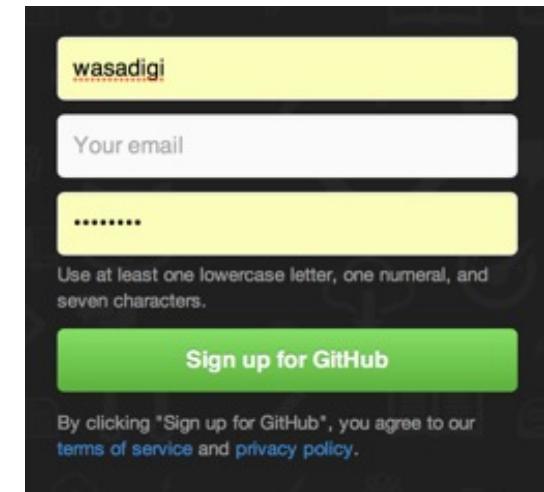
```
class BartenderTest {  
  
    @Test  
    void aBartenderShouldReplyToGreetings() {  
        Bartender john = new Bartender();  
        GreetingsRequest request = new GreetingsRequest("Hi there");  
        GreetingsResponse response = john.greet(request);  
        assertEquals("hello, how can I help you?", response.getText());  
    }  
  
}
```

```
public class Bartender {  
  
    public GreetingsResponse greet(GreetingsRequest request) {  
        return new GreetingsResponse("hello, how can I help you?");  
    }  
  
    ...  
}
```

```
import lombok.Data;  
  
@Data  
public class GreetingsRequest implements IMessage {  
  
    private final String text;  
  
}
```

# GitHub Setup

- **Sign up** for GitHub and get your own account:
  - Go to <http://www.github.com>
- Add your **SSH key**:
  - Go to your accounts settings. You will find an option to manage your SSH keys.
  - If you don't have a SSH key yet, follow the instructions in the online help.
  - If you are using windows, you will need to use Git BASH.
- **Create** your first repo, hosted on Github.
- **Copy the SSH URL** of the repo.

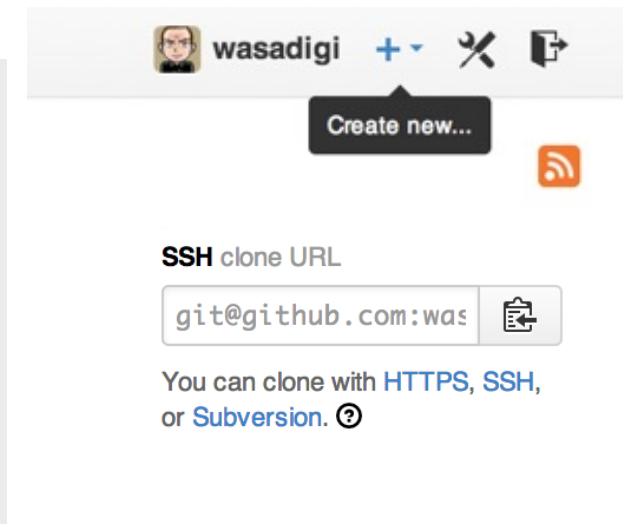
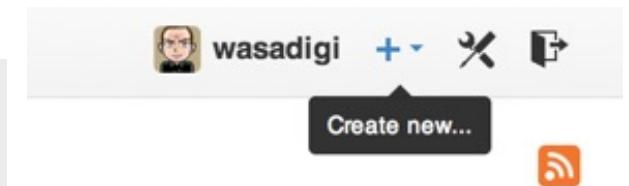
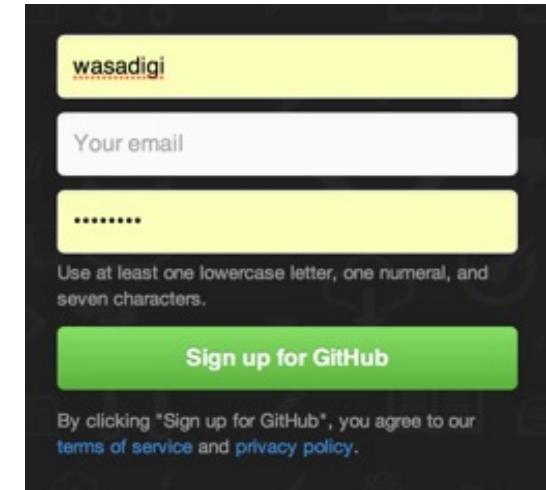


# Validate your setup

- **Clone your repo to your laptop**

- Open a **terminal window** (Terminal on Mac OS, Git BASH on Windows, etc.)
- Create a **new directory** to host your clone of the repo and get into it.
- **Clone** the repo, using the **SSH URL**.
- **Create** a file, **add** it to the staging area, **commit** the changes and finally **push** the commit Github.

```
$ mkdir myspace
$ cd myspace
$ git clone git@github.com:UUUUU/RRRRR.git
$ git touch firstFile.txt
$ git add firstFile.txt
$ git commit -m "I have added my first file"
$ git push
```



If you do this, you will have YOUR  
clone of MY repo hosted on Github

# Forks & Clones

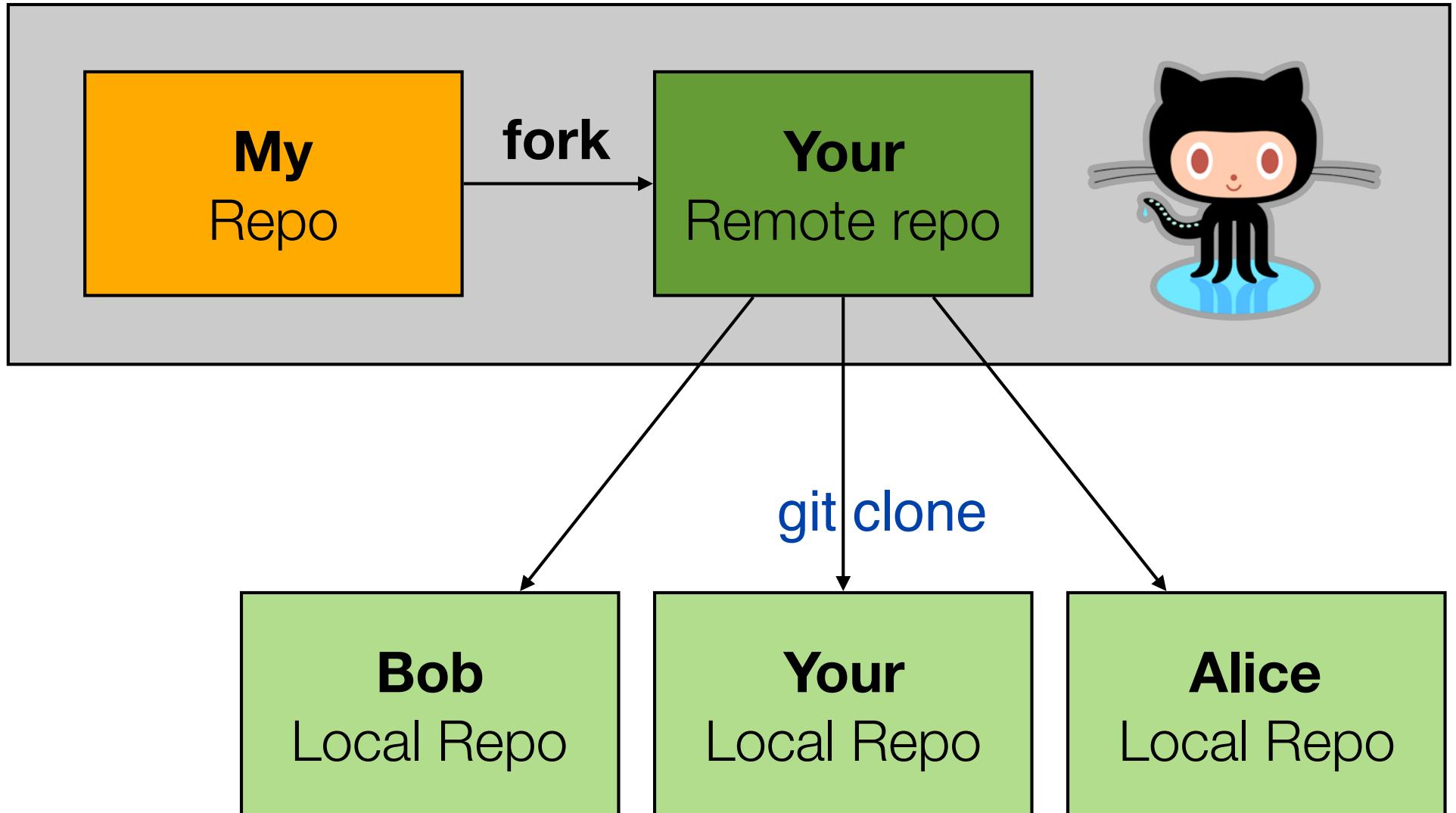
The screenshot shows a GitHub repository page for 'wasadigi/Teaching-COMEM-MWS'. The repository is public and has 6 commits, 2 branches, 0 releases, and 1 contributor. The 'master' branch is selected. The README.md file contains the following text:

```
Welcome to the Mobile Web Services  
(MWS) Git Repository
```

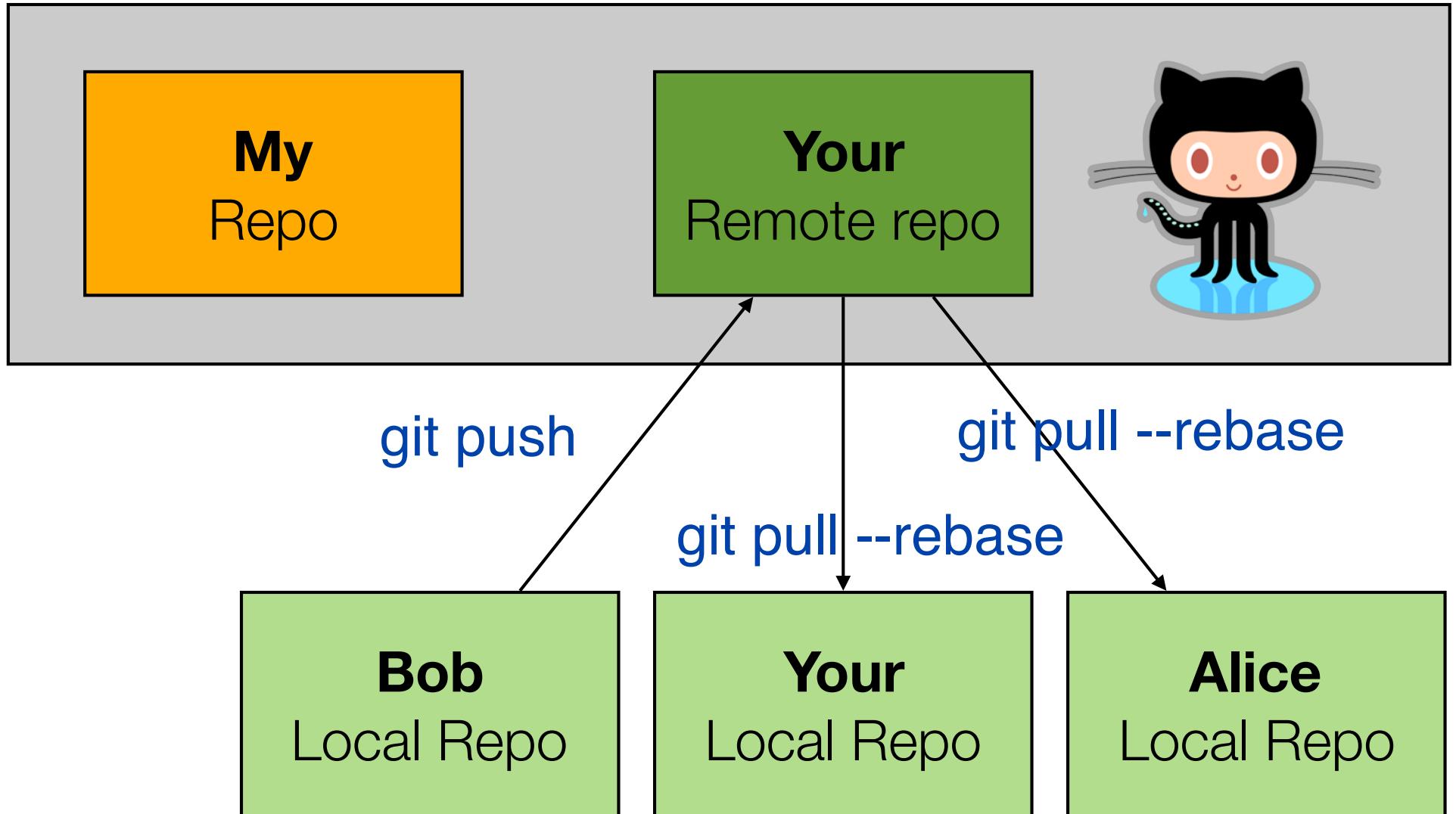
The repository description states: "This repository is used for the Mobile Web Services course, organized at the University of Applied Sciences of Western Switzerland in the COMEM Department."

On the right side of the page, there is a sidebar with various repository links: Code, Issues, Pull Requests, Wiki, Pulse, Graphs, Network, and Settings. Below the sidebar, there is an SSH clone URL field and download options for 'Clone in Desktop' and 'Download ZIP'.

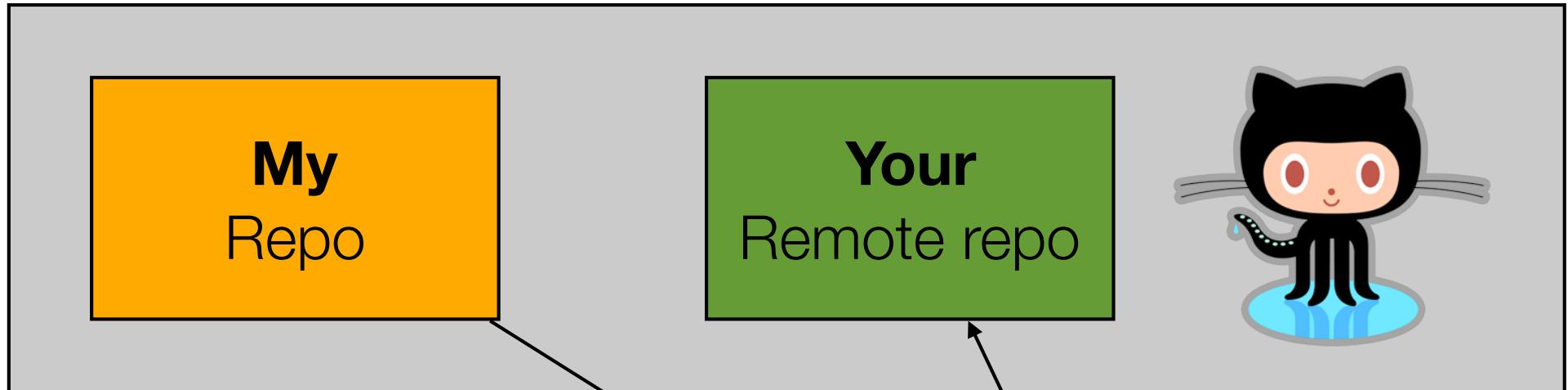
# Forking My Repo on GitHub



# Forking My Repo on GitHub



# How Do Will You Get **My** Updates?



```
# configuration, do it once
git remote add upstream git://github.com/UU/RR.git
git remote -v
```

**git fetch upstream**  
**git rebase upstream/master**

**Your**  
**Local Repo**

# Tools



# Git & GitHub

- **Step 1: install Git**
  - <http://git-scm.com/downloads>
  - <http://git-scm.com/book/en/Getting-Started-Installing-Git>
  - Check point: are you able to invoke the git command from the shell?
- **Step 2: configure Git**
  - <https://help.github.com/articles/set-up-git>
- **Step 3: configure SSH**
  - <http://guides.beanstalkapp.com/version-control/git-on-windows.html#installing-ssh-keys>



# Using Git locally

```
$ mkdir my-project
$ cd my-project
$ git init
$ ls -al
```

- You do not **have to use a server**: Git is already useful to manage versions of your files on your local machine.
- The **git init** command creates a **local repository**. If you look carefully, you will see a **hidden .git directory**, where Git keeps all of his data.
- **Important:** your **my-project** directory is your **working directory**. If you simply create files in it, they will not immediately be part of your repository!

# Using Git locally

```
$ echo "text a" > a.txt
$ git status
$ git add a.txt
$ git commit -m "First version of a.txt"
$ echo "my mod on text a" > a.txt
$ git status
```

- A **commit** is a **snapshot** of your repository. Git maintains a **graph of commits** and you can always **recover the state** of a particular commit.
- When **you have modified files in your working directory**, you need to specify which ones should be **part of the next commit**.
- You use the **git add** command to add a file to the so-called **staging area**. It will be part of the next commit.
- You use the **git status** command to **check the content** of your working directory and of your staging area.

**More info:** <http://git-scm.com/book/en/Git-Basics-Recording-Changes-to-the-Repository>

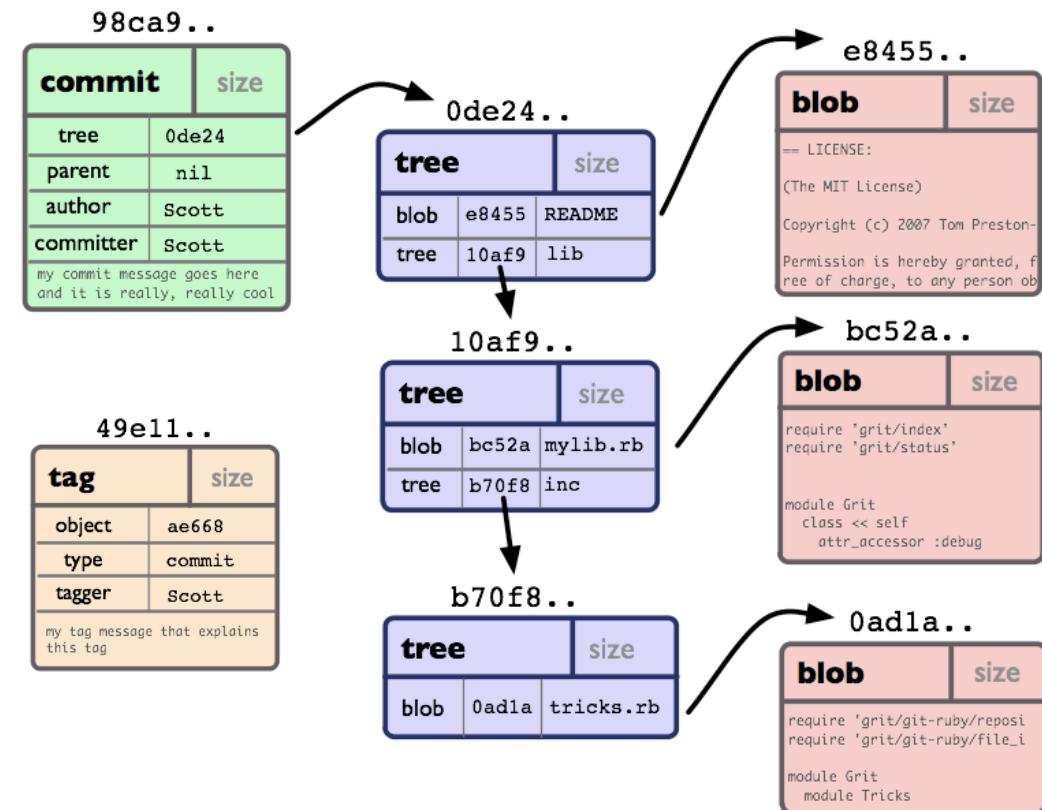
# Using Git locally



- **Git is a "content-addressable file system"**
- **Git uses a key-value store:**
  - When you store a file in a repo, git computes a SHA-1 hash of its content.
  - The hash is used as a key to index the file in the store.
  - For this reason, two files with exactly the same content are stored only once in the git repository.
  - **Go in the .git hidden directory, have a look at the ./objects directory and you will find this key-value store.**

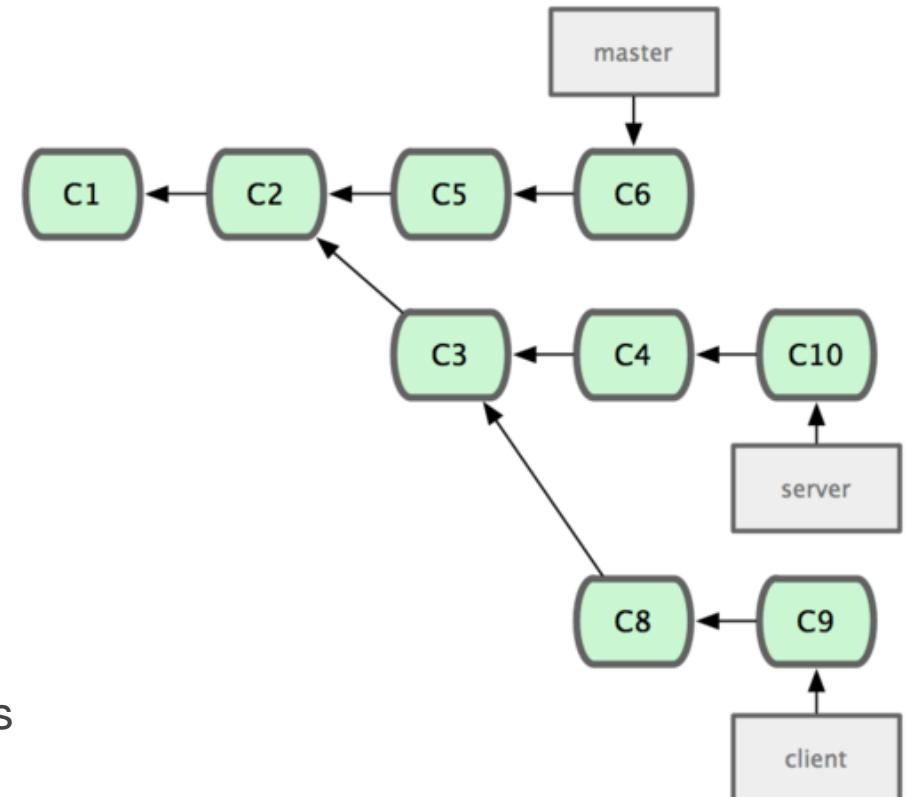
# The Git Object Model

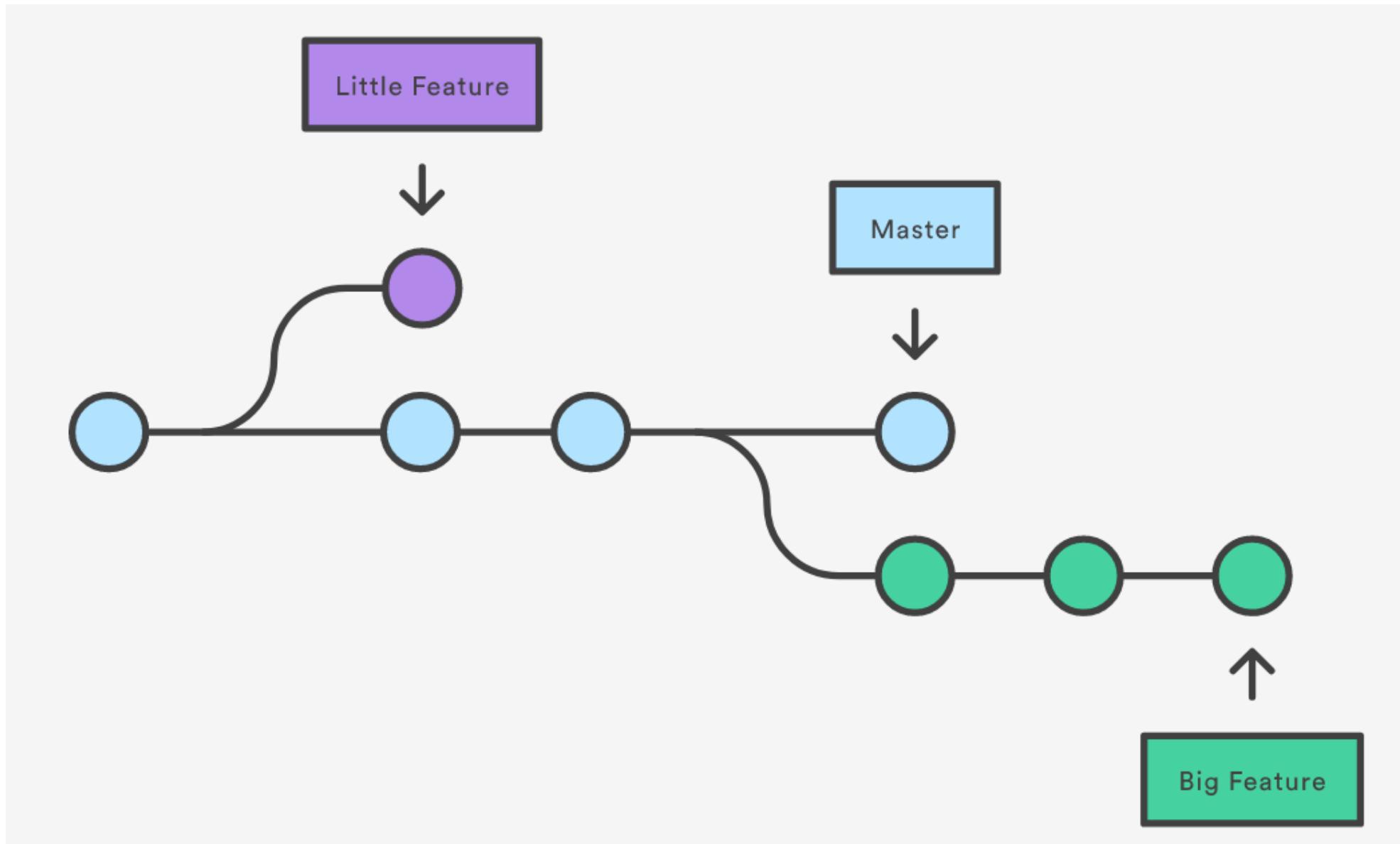
- **BLOBs** store files.
  - **Trees** store **directory structures** in the file system. A tree has pointers to subtrees and BLOBs.
  - **Commits** identify snapshots in the history. **Every commit points to a version of the top-level tree**. A commit also stores metadata: author, parent, message, etc.
  - **Annotated Tags** (created with git tag -a) are used to mark releases.
  - Recommendation: do not use lightweight tags (created with git tag, without -a)



# The Git history is a DAG structure

- **A DAG is a Directed Acyclic Graph**
  - Except for the initial commit, every commit has at least one parent.
  - Commits with more than one parent are "merge" commits.
- **In this example:**
  - C1 is the initial commit
  - There are no merge commits
  - master, server and client are branches
  - The branches have never been merged
  - The last commit on the client branch is C9
  - All branches have C1 and C2 as common ancestors





<https://www.atlassian.com/git/tutorials/using-branches>

# Git branches are only pointers

- Branches are stored in files located in the `.git/refs/heads` directory.

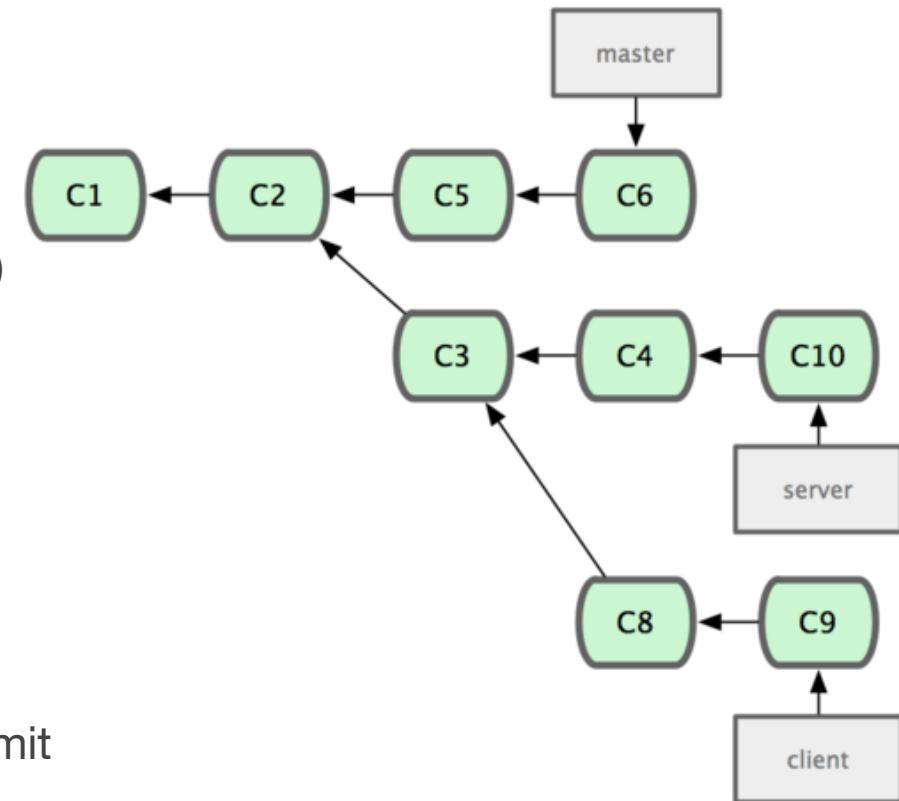
- **Each branch is stored in a file, which only contains the hash of a commit.**

- **What is HEAD?**

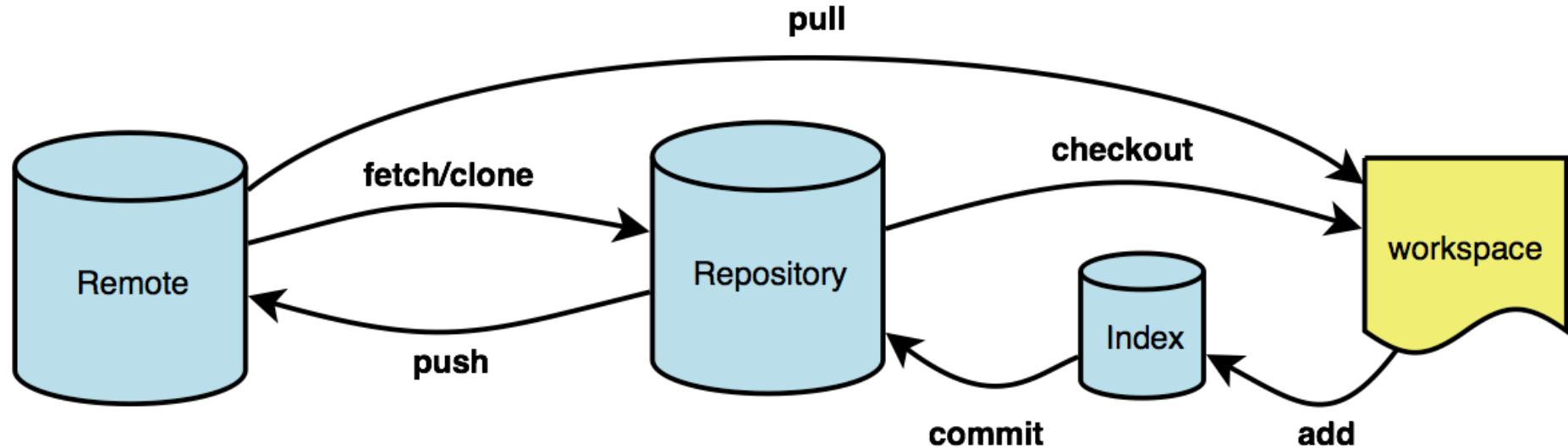
- HEAD means "the tip of the current branch" (last commit of the current branch)
- When you checkout a branch, HEAD is the name of this branch.

- **What is  $\text{HEAD}^n$ ?**

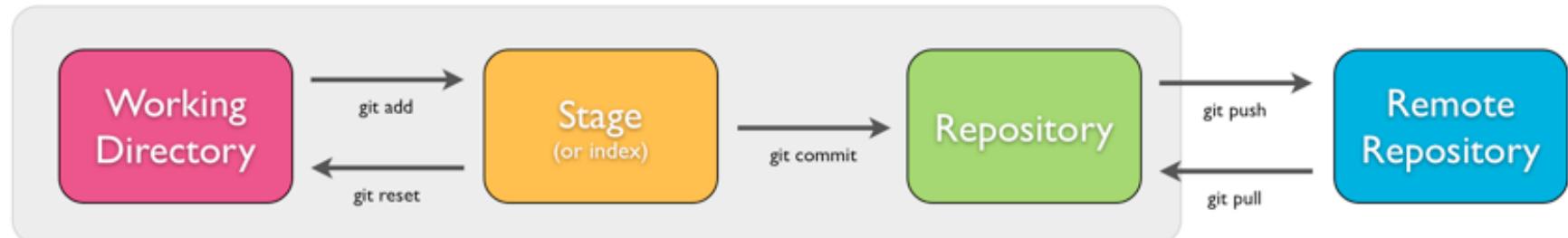
- It is a reference to the nth parent of HEAD
- $\text{HEAD}^1$  means the parent of the last commit
- $\text{HEAD}^2$  means its grand-parent



# Git & Remote Repositories



**Source:** <http://illustrated-git.readthedocs.org/en/latest/>



**Source:** <http://bramus.github.io/ws2-sws-course-materials/xx.git.html#/4/1>