Advanced Operating Systems
Lab Tools Intro

2017/18
www.ce.unican.es
vpuente@unican.es

Environment

- (Some) current tools as a “regular” user
  - Virtual Machine deployment/provisioning system (vagrant)
  - Version control system (git)

Vagrant

- It's a abstraction layer on top of any virtualization layer
  - Works with VirtualBox, Vmware, Hyper-V, …, lxc, docker, etc…
  - Although the lab work can be done with plain VirtualBox, vagrant is recommended to avoid the hassle of crude virtual machines (VM)
  - Used as a "simplification" tool for VM handling

- Recipe
  1. Install vagrant and VirtualBox (Linux/Windows/OSX)
  2. `mkdir myWorkingDir`
  3. `vagrant init debian/jessie64`
  4. `vagrant up`
  5. `vagrant ssh !`

How to work with vagrant

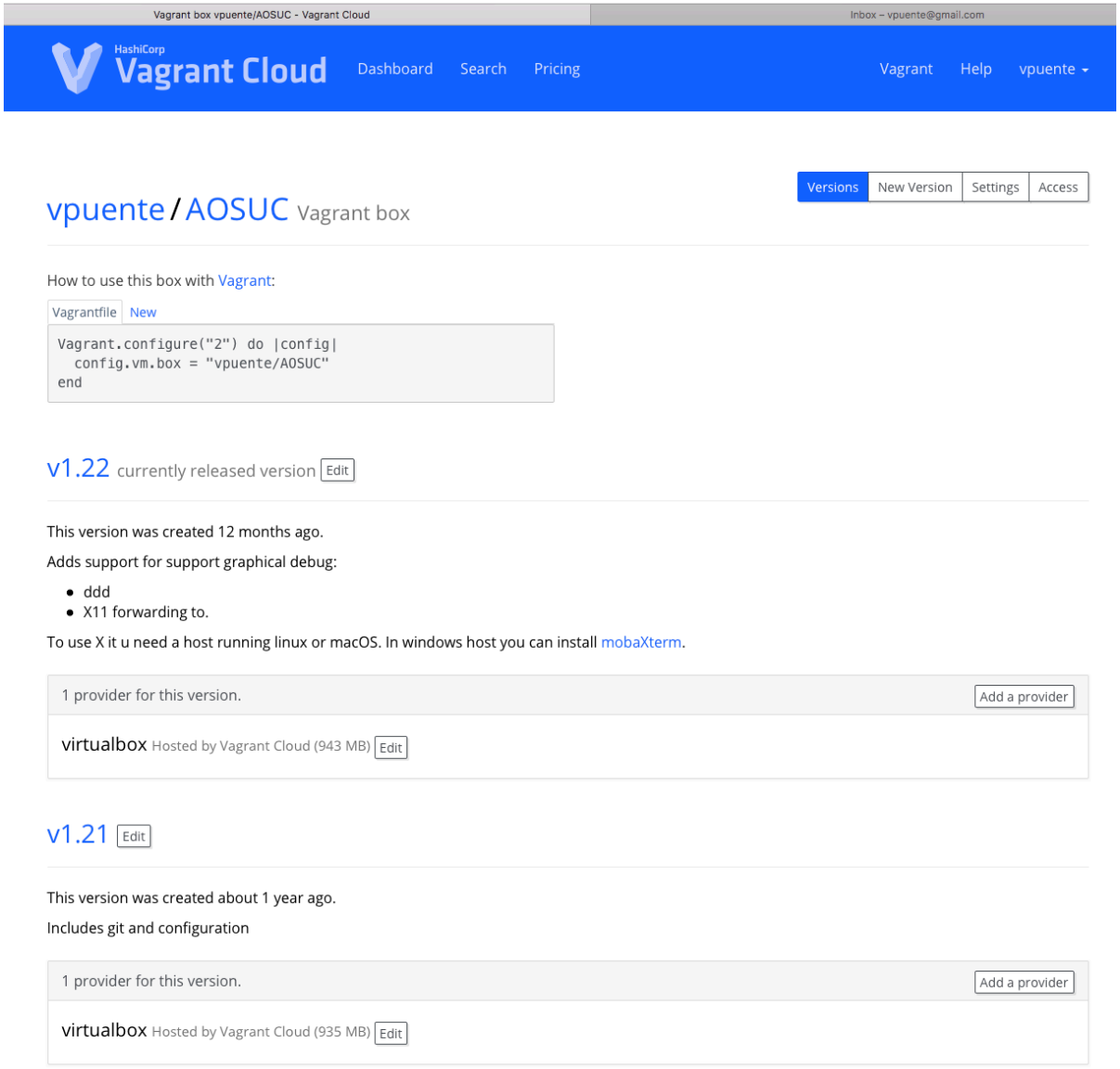
- Boxes != instances
- List boxes
  - `vagrant box list`
- Create a new instance (persistent)
  - `mkdir directory; cd directory; Vagrant init some_box`
- File interaction
  - Use the shared dir and work in the host
    - `/vagrant` directory is working directory in host
    - Requires VirtualBox utils installed in the VM (version number should match)
    - Beware clock-skew between VM and host (might affect make/git)
  - Use x11 forwarding (required a X11 server)
  - Use rdp (requires Windows host and a rDesktop/VNC client)
  - Use VirtualBox interface

Other useful commands

- To handle instances
  - To inspect the system status
    - `vagrant global-status`
  - To delete instances use "`vagrant destroy`" (never delete a VM from virtualBox interface!)

- To handle boxes
  - `vagrant box list` -- List versions installed, that might me updated in remote server
  - `vagrant box update` -- To download updated versions of the box (v.gr. if the box is upgraded on atlas.hashicorp.com)
  - `vagrant box remove vpuente/AOSUC1617` --box-version 1.1 -- cleans old version for that box

AOS Vagrant Box: vpuente/AOSUC1617 (Demo)



Advantages

- Allow to provision the Environment without tinkering with the files
  - Download the box "once"! (not each time you install it)
  - Use it many times in many different contexts (v.gr. a particular lab or section)
    - Share the changes (labs, exams, etc...)
- Allows to run the box to any provider (beyond VirtualBox)
  - Local or external (i.e. Cloud provider such as AWS, GCE, Azure, etc...)
  - A higher level of automation are Chef and Puppet (automated delivery and provisioning)
  - Plays nice with Docker, CI systems (github, gitlab), ...
- Only used as a tool!
  - Interesting in learning more?: Sistemas Virtualización y Seguridad

There are other options?

- It's possible to do the lab natively in:
  - Linux (ugh)
  - Windows 10
    - Using linux subsystem
    - https://msdn.microsoft.com/es-es/commandline/wsl/install_guide
  - osX
    - Using port or brew
    - https://stackoverflow.com/questions/39052271/compile-xv6-on-mac
- My advice?
  - Pick your choice…

Others

- Code editor
  - Vim ☺
  - **Visual Source Code**
  - SublimeText
  - Atom
  - …
- Debugger
  - Remote-debug (to QEMU)
  - gdb http://beej.us/guide/bggdb/
  - Other gdb frontends (ddd, VSC, etc…)

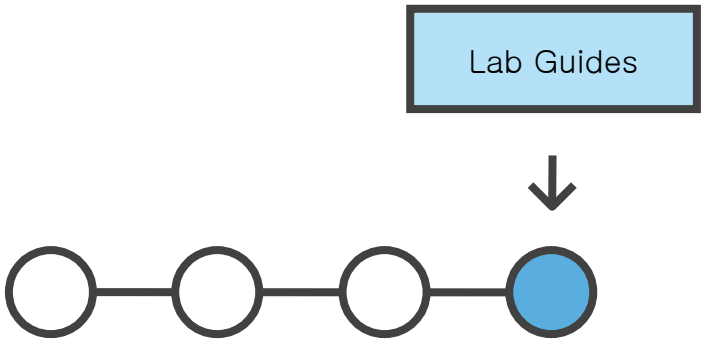
Lab work

- All lab material will be available in git lab repository
  - `git clone https://gitlab.com/AOSUC/Lab.git`
- Material
  - Where you have to work
- Reference
  - OSTEP original material (ostep.org)

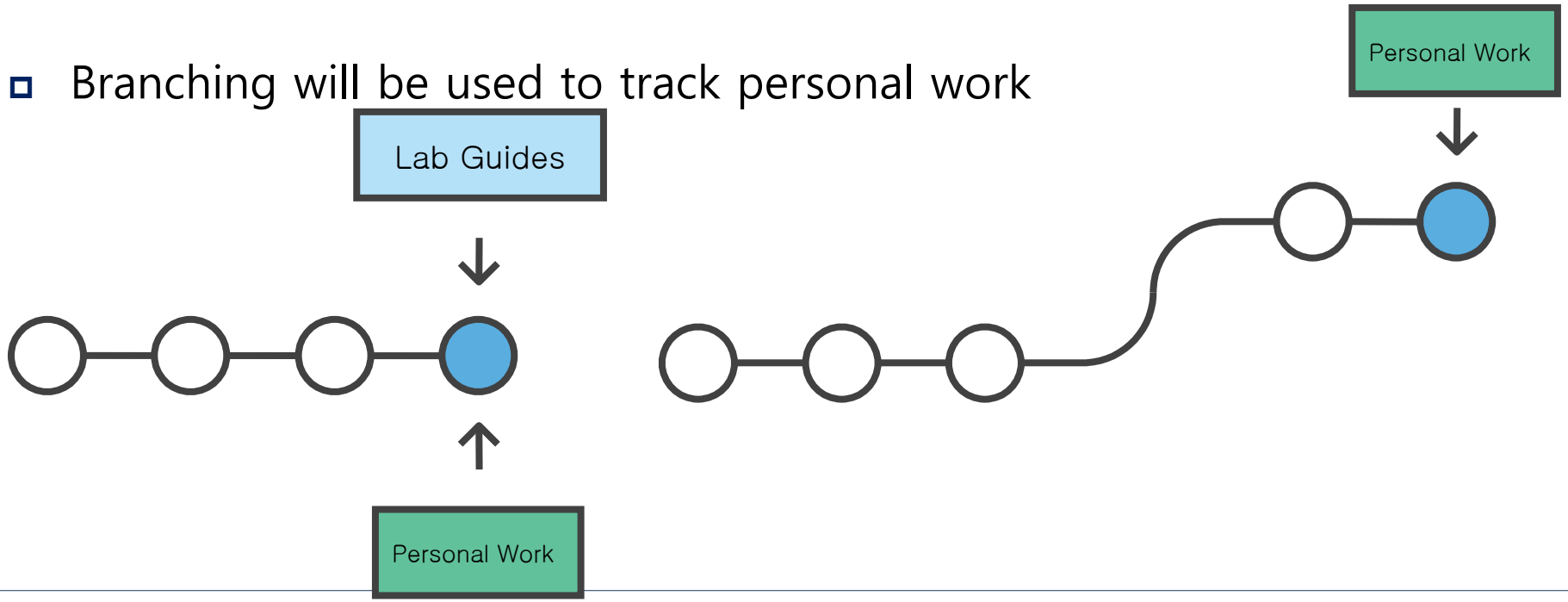
- We will work using a simple git branch+up-stream based workflow
  - https://www.atlassian.com/git/tutorials/what-is-git

Personal Work tracking

- Git repo contains a “chain” of atomic changes called commits



- Branching will be used to track personal work

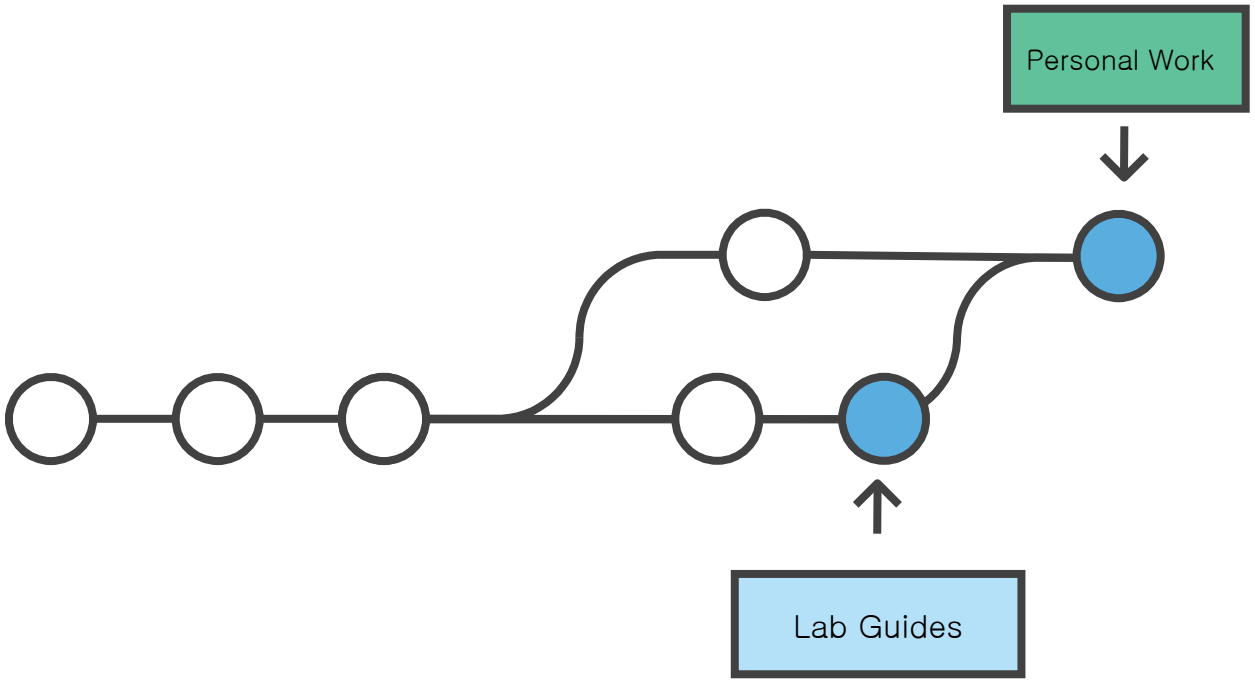


Workflow

- Create an account in gitlab.com
- Create a private repository (for lab)
- Fork https://gitlab.com/AOSUC/Lab.git
- Invite vpuente@gmail.com to such repository
- Start working
  - `git clone https://username@gitlab.com/Lab.git`
  - `git branch <DNI>`
  - `git checkout <DNI>`
  - START WORKING THERE / COMMIT AS MUCH AS U NEED!!!
    - `git add NEW_FILES` **`DO NOT MODIFY ALREADY PRESENT FILES (merge conflicts)`**
    - **`<WORK>`**
    - `git commit` **`<WORK>`** `git commit` **`<WORK>`** `git commit` **`<WORK>`** **…**
    - `git push`
  - git pull https://username@gitlab.com/AOSUC/Lab.git

Updating of Guides (Pull)

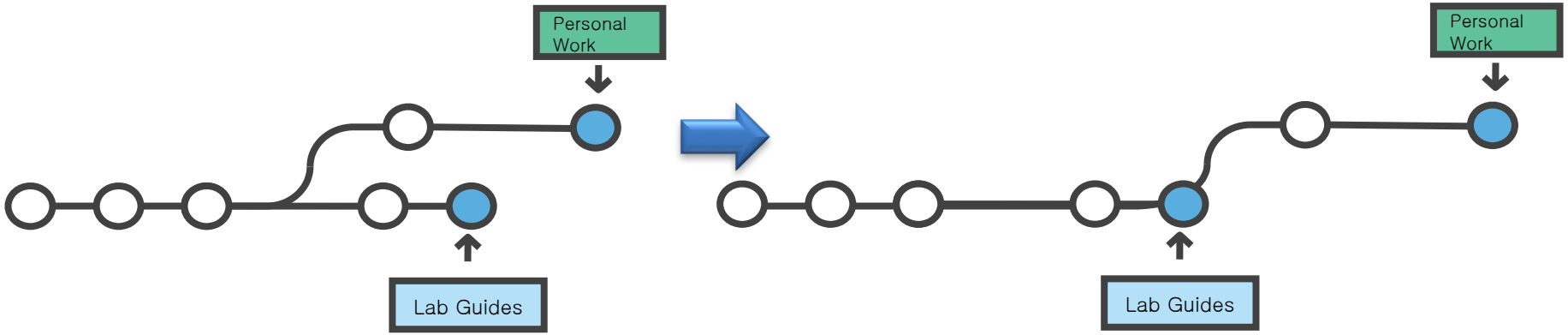
- Merging updates in material



- Allow the professor to track your work
- Allow to automatize C&P detection

Download changes from the common repo

- How to integrate changes from mainline in my fork?
  - `git checkout master`
  - `git pull https://gitlab.com/AOSUC/Lab`
  - `git checkout <DNI>`
  - `git rebase master`

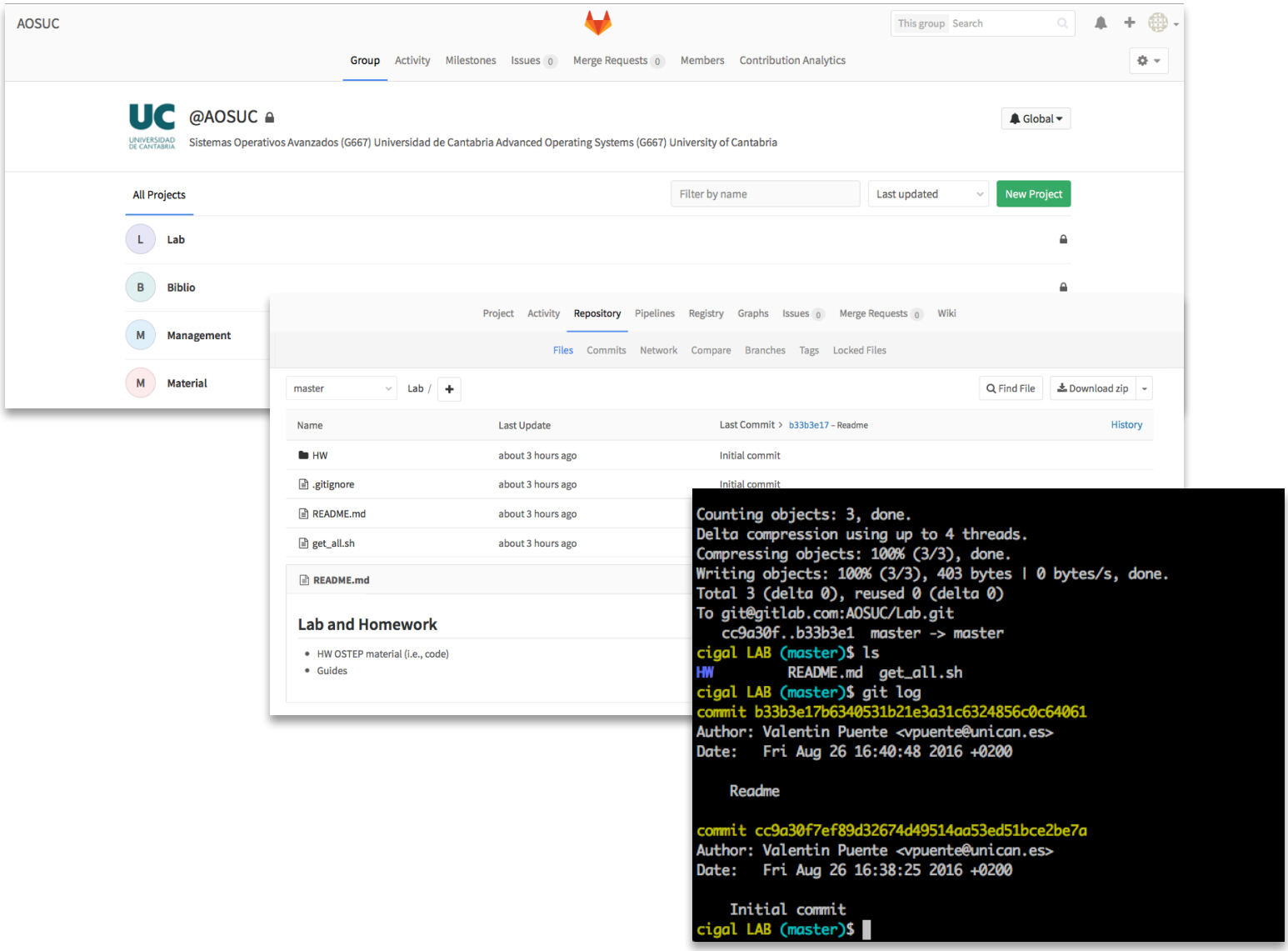


- This process might fail if done in “non” reliable clock environment
  - For example if we have clock-skew between system and files (v.gr. Virtual Box and host shared directory !!!)

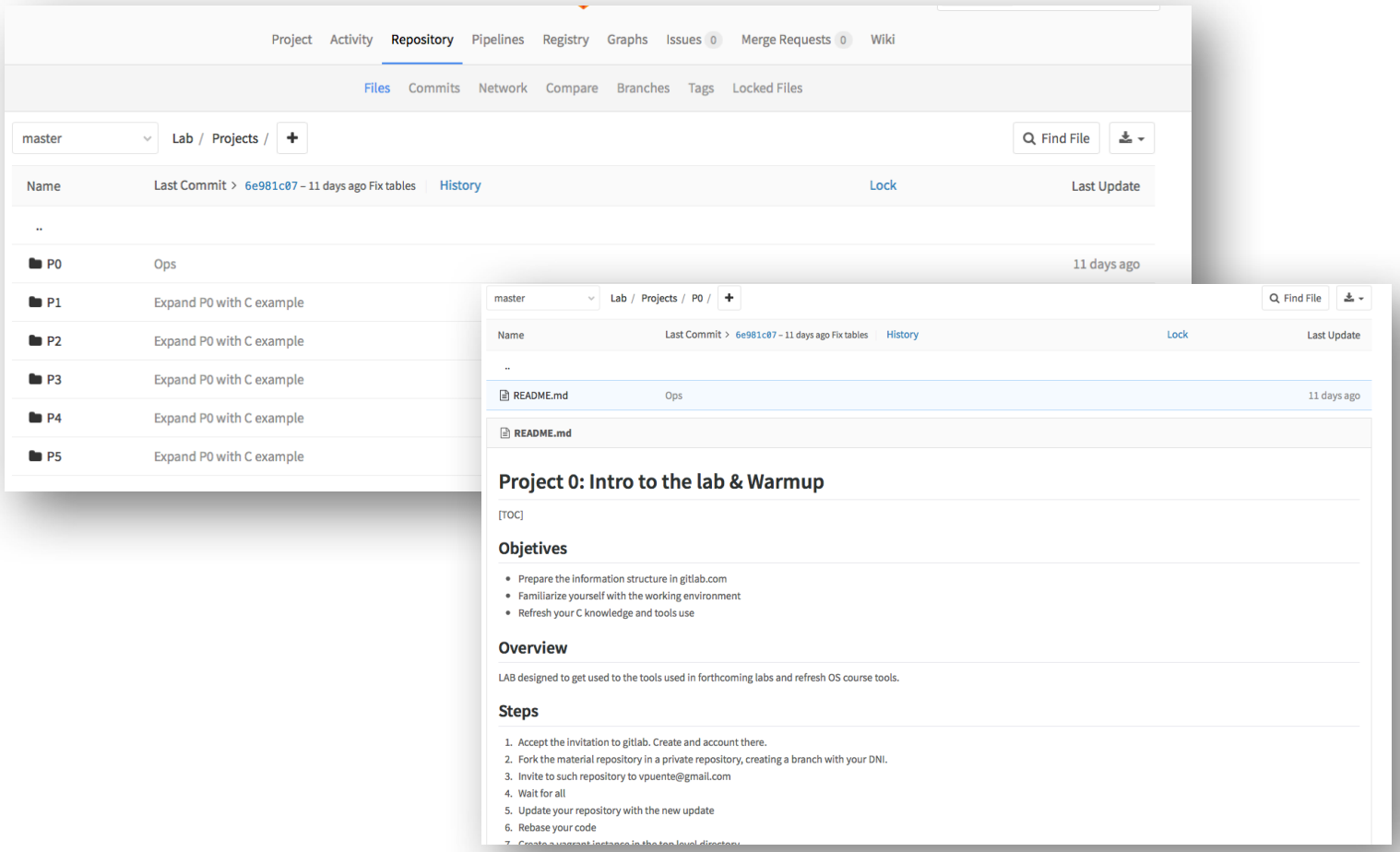
Next level (not required but advised to try at least)

- Create additional branches for a particular work
- Merge with DNI branch when done
  - Only use the Web interface to merge
  - Command line tool are not easy to use (especially when conflicts appears)
- Allows to work on multiple tasks at once (and do not have a chaos in hands)
- Use issue board to track your progress

Gitlab Interface
(not needed git deep understanding)

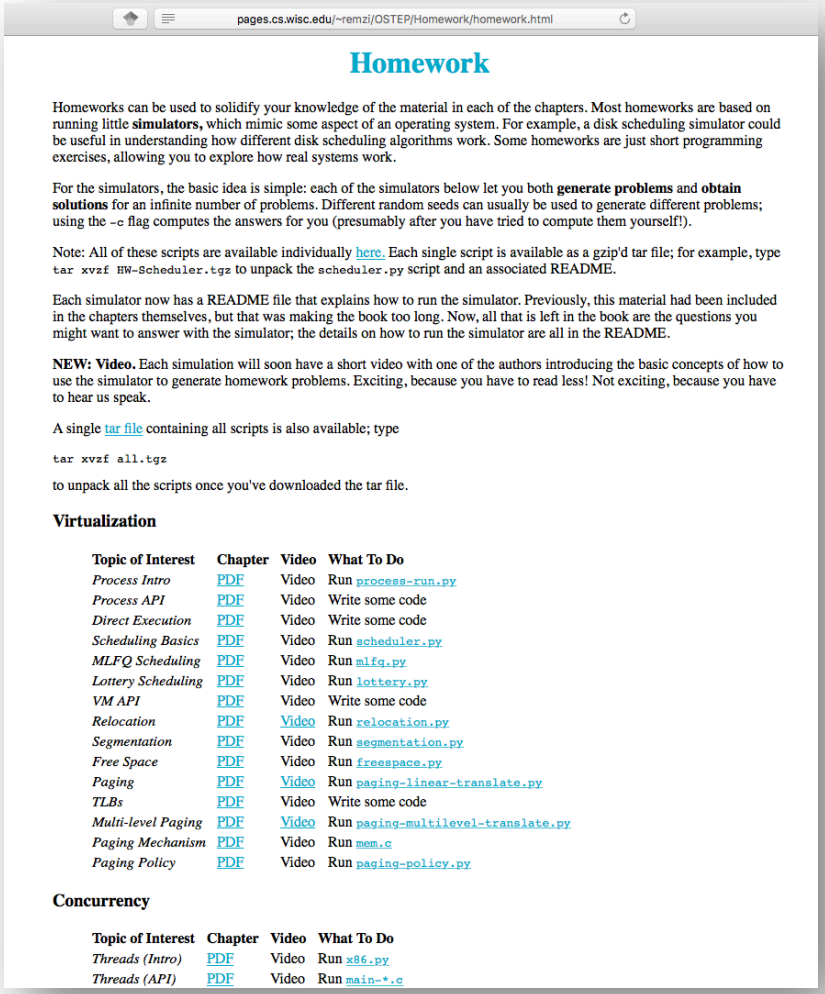


GitLab interface (Demo)



Structure of the repository (Demo)

- Homework's
  - Book material
- Projects
  - Develop here. Add a SOLUTION.md at the end on the P{$$} dir.
- xv6
  - Source code if the hacking environment used



pages.cs.wisc.edu/~remzi/OSTEP/Homework/homework.html

Homework

Homeworks can be used to solidify your knowledge of the material in each of the chapters. Most homeworks are based on running little **simulators,** which mimic some aspect of an operating system. For example, a disk scheduling simulator could be useful in understanding how different disk scheduling algorithms work. Some homeworks are just short programming exercises, allowing you to explore how real systems work.

For the simulators, the basic idea is simple: each of the simulators below let you both **generate problems** and **obtain solutions** for an infinite number of problems. Different random seeds can usually be used to generate different problems; using the `-c` flag computes the answers for you (presumably after you have tried to compute them yourself!).

Note: All of these scripts are available individually here. Each single script is available as a gzip'd tar file; for example, type `tar xvzf HW-Scheduler.tgz` to unpack the `scheduler.py` script and an associated README.

Each simulator now has a README file that explains how to run the simulator. Previously, this material had been included in the chapters themselves, but that was making the book too long. Now, all that is left in the book are the questions you might want to answer with the simulator; the details on how to run the simulator are all in the README.

NEW: Video. Each simulation will soon have a short video with one of the authors introducing the basic concepts of how to use the simulator to generate homework problems. Exciting, because you have to read less! Not exciting, because you have to hear us speak.

A single tar file containing all scripts is also available; type

```
tar xvzf all.tgz
```

to unpack all the scripts once you've downloaded the tar file.

Virtualization

Topic of Interest	Chapter	Video	What To Do
Process Intro	PDF	Video	Run `process-run.py`
Process API	PDF	Video	Write some code
Direct Execution	PDF	Video	Write some code
Scheduling Basics	PDF	Video	Run `scheduler.py`
MLFQ Scheduling	PDF	Video	Run `mlfq.py`
Lottery Scheduling	PDF	Video	Run `lottery.py`
VM API	PDF	Video	Write some code
Relocation	PDF	Video	Run `relocation.py`
Segmentation	PDF	Video	Run `segmentation.py`
Free Space	PDF	Video	Run `freespace.py`
Paging	PDF	Video	Run `paging-linear-translate.py`
TLBs	PDF	Video	Write some code
Multi-level Paging	PDF	Video	Run `paging-multilevel-translate.py`
Paging Mechanism	PDF	Video	Run `mem.c`
Paging Policy	PDF	Video	Run `paging-policy.py`

Concurrency

Topic of Interest	Chapter	Video	What To Do
Threads (Intro)	PDF	Video	Run `x86.py`
Threads (API)	PDF	Video	Run `main-*.c`