

School of Informatics, Computing,  
and Cyber Systems

# INF502

## SOFTWARE DEVELOPMENT METHODOLOGIES

Lecture: Git introduction

**Dr. Igor Steinmacher**

e-mail: [Igor.Steinmacher@nau.edu](mailto:Igor.Steinmacher@nau.edu)

Twitter: @igorsteinmacher

# CODE MANAGEMENT/VERSIONING

- Team development
  - Code sharing and versioning...



# CODE MANAGEMENT/VERSIONING

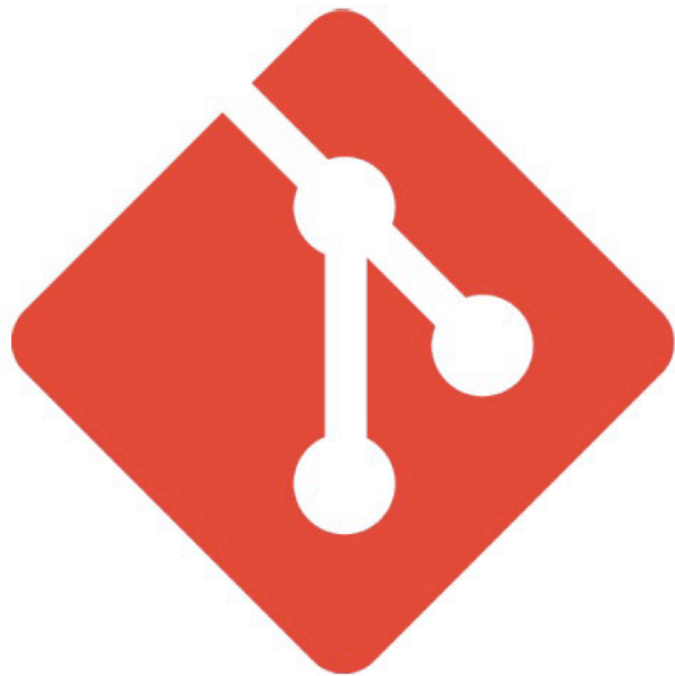


**CVS**



...

**WE WILL FOCUS ON:**



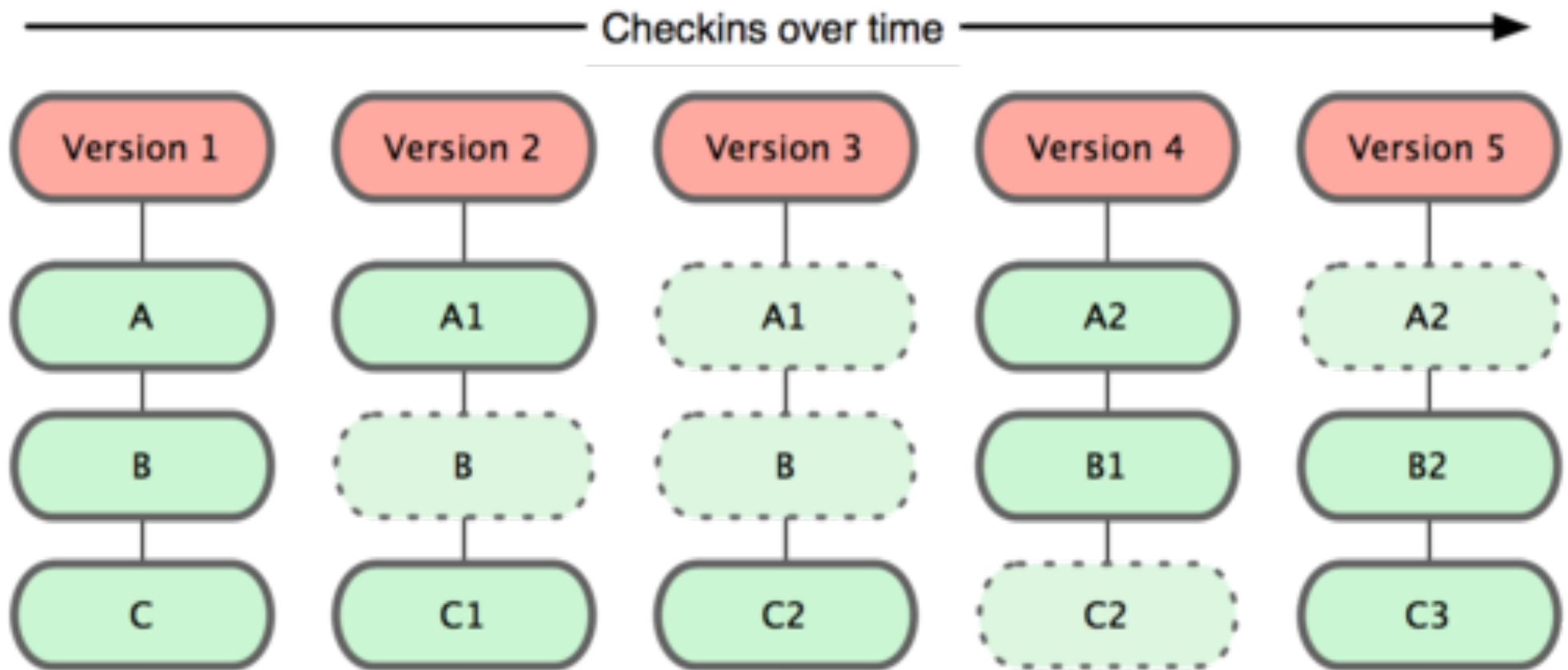
**git**

# WHO OFFERS THIS SERVICE

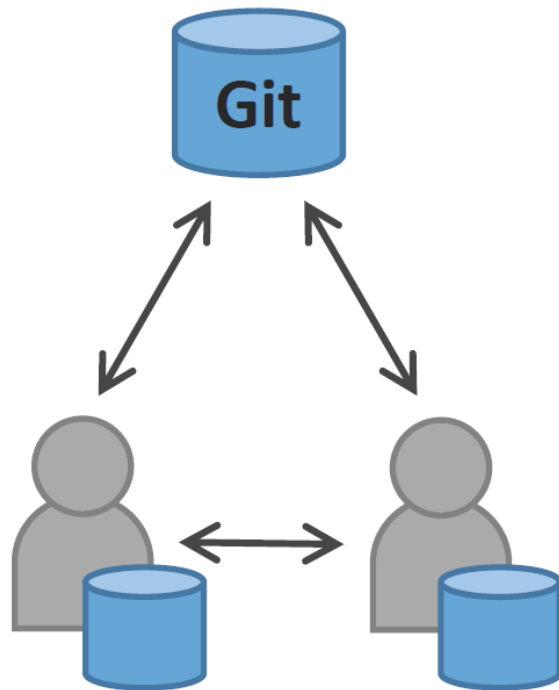
- Your machine! (?)



# HOW GIT MANAGES FILES OVER TIME

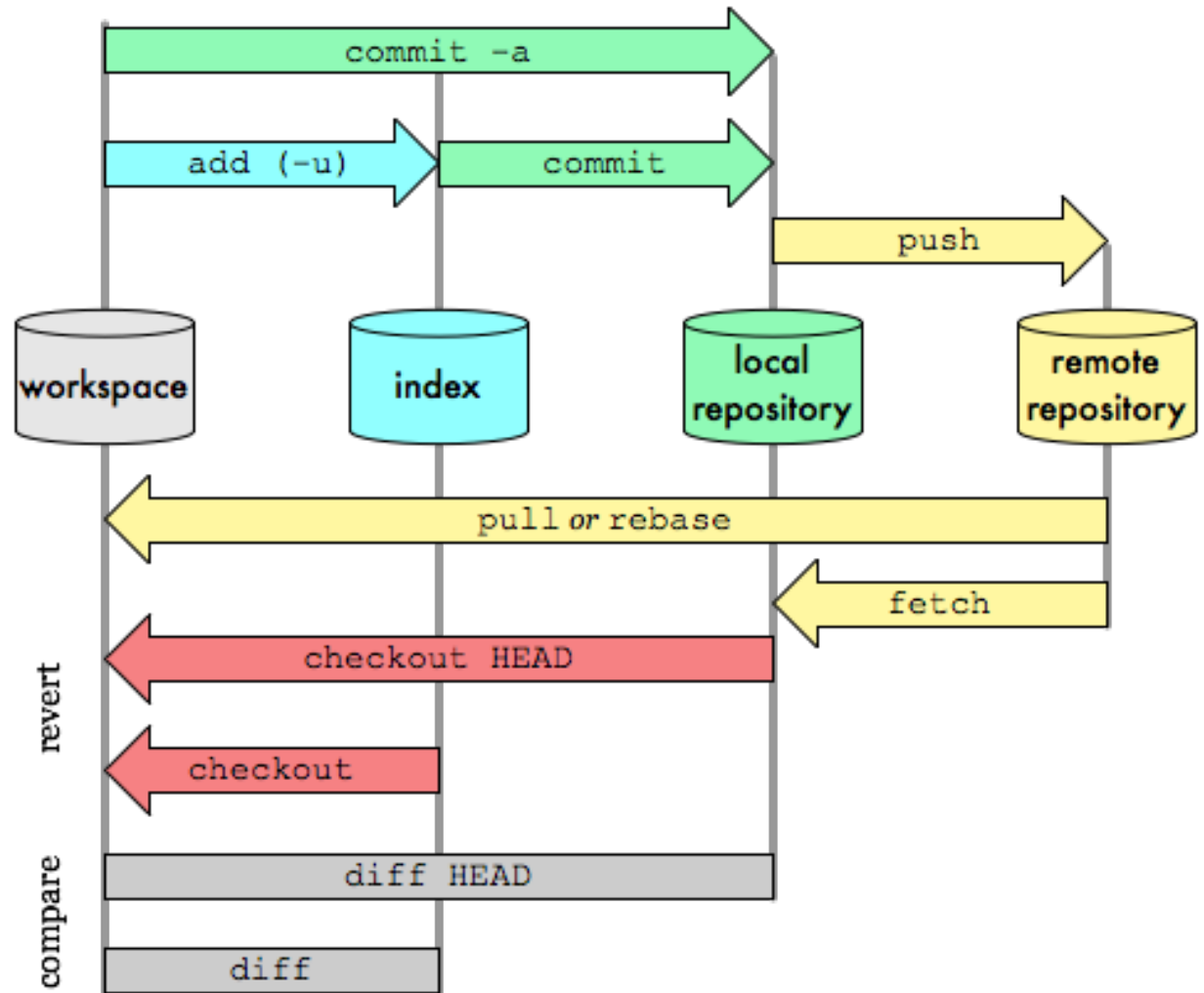


# GIT - OVERALL

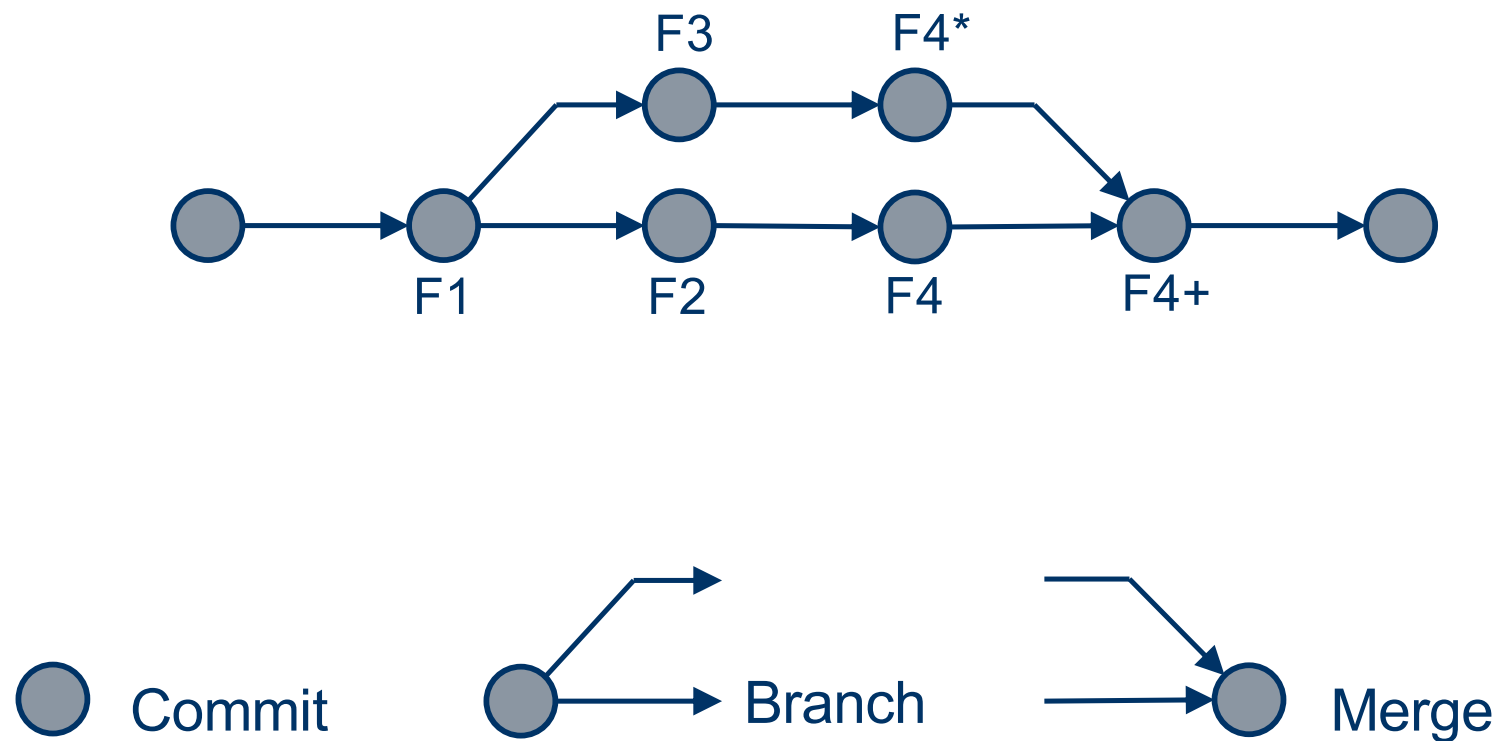


## Git Data Transport Commands

<http://osteele.com>

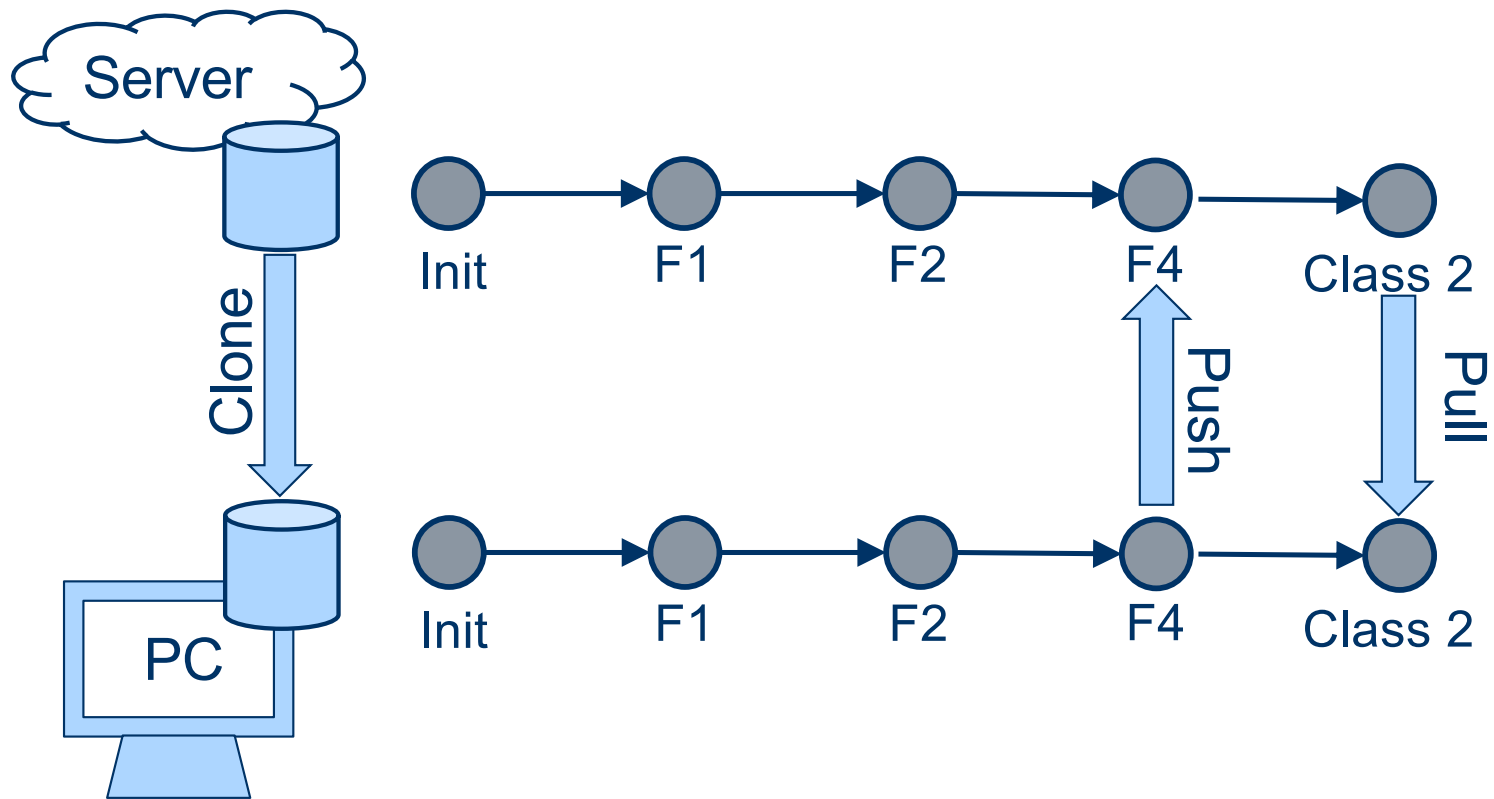


# GIT LOCAL FLOW - EXAMPLE





# GIT LOCAL FLOW - EXAMPLE



# IT'S HANDS ON TIME

- 2 Moments
- Moment 1:
  - **Local commands: add, commit, branch, merge, conflicts..**
- Moment 2
  - **Interaction with the remote repo: Push, pull**

# KICKING OFF

- git config --global user.name "Igor Steinmacher"

For all repos



- git config --global user.email "igor.Steinmacher@nau.edu"

- Create a folder / access this folder
- git init
  - **This folder is a repo**

# HANDS ON

- Create a file
- Check the status of the repo
  - `git status`
- Add the file to the index
  - `git add <filename>`
- Check the status

# HANDS ON

- Our first commit
  - **git -a -m “Our first commit!!!”**
    - -a → all files
    - -m → will include a commit message
- Check the last commits
  - **git log**
- Check what has been done in the last commit
  - **git show**

# HANDS ON

- Let's!
  - Change the file
  - `git status`
  - `git commit ...`
  - `git status`
  - `git log`
- And this is the basic flow to put your contributions back to the repo
  - `add`
  - `commit`
  - `status`
  - `log`
  - `show`

# LET'S BRANCH IT OUT!

- Listing your branches
  - `git branch`
- Create a branch
  - `git branch <NEW_BRANCH>`
- Use another branch
  - `git checkout <BRANCH_NAME>`
- Latest two in one command
  - `git checkout -b <NEW_BRANCH>`

# WORKING IN A NEW BRANCH

- `git checkout -b branchNew`
- `<Change one existing file here>`
  - **“Hi, my name is Hugh.”**
- `git commit -a -m “Introducing myself”`
- `<Check the content of the file>`
  
- `git checkout master`
- `<Check the content of the file>`
  - **What?!?!**



# UPDATING THE MASTER BRANCH

- Usually we branch out for versions, features, bug fixes...
  - Later on merging back to master
- How to merge our recently changed file, then?
  - In the master branch:
  - `git merge <other_branch>`
- If everything goes smooth... sweet

# DEALING WITH SMALL CONFLICTS

- Imagine if you change a file in your branch, and someone else changed the same file
  - **CONFLICT!!!!**
- Can we still merge it?!?!?
  - **Let's see:**
    - Change branch
    - Change file
    - Commit
    - Back to master
    - Change the same file
    - Commit
    - MERGE!

Auto-merging <file>  
CONFLICT (content): Merge conflict in <file>  
Automatic merge failed; fix conflicts and then  
commit the result.

# USUALLY... FOR THE EASY ONES

Here comes common text before  
the area where the conflict happens  
and bla bla bla

<<<<<<< HEAD

This is what was in the  
master branch

=====

And this...  
was in the other branch

>>>>>>> other branch

More text that was common,  
and no conflict happened here

# EXERCISING IT OUT

- Make a new branch called bugfix
- Checkout the bugfix branch with `git checkout bugfix`
- Create/change a file and commit
- Go back to master with `git checkout`
- Change/create a file (different from the previous one) and commit again
- Merge the branch bugfix into master with `git merge`

# REBASING IT ALL!!!

- Another way of combining branches
- We can take a set of commits, and copy them in another branch
- Master and our branch are not sync'ed
  - **Commit made in master**
  - **Commit made in the branch**
- From the branch, we can
  - **git rebase master**
  - **Now the index of the master is “outdated” → HEAD is pointing to bugfix last commit**
    - `git log --graph --all`

# MOVING FROM HERE TO THERE

- HEAD is the pointer name for the last checked out commit
- We can “detach” the head by “checking out” a specific commit
  - **git checkout <commit SHA>**
- This is not “safe”
  - **git checkout <branch>**
    - To get back
- Moving in the commit tree
  - **Moving one commit at a time with ^**
    - git checkout HEAD^
    - git checkout master^
  - **Moving a number of times with ~<num>**
    - git checkout master~3

# MOVING FROM HERE TO THERE

- We can move a branch!
  - `git branch -f <BRANCH> HEAD~3`
- We can also revert changes
  - `git reset HEAD~2`
    - Move the current branch to HEAD - 2 commits position
    - Works LOCAL
    - `git reflog` → see previous commits
    - `git reset <SHA>` → SHA of the commit before the reset for “unresetting”
  - `git revert HEAD` → To reverse changes to send upstream

# CHERRY-PICKING

- Use when you don't want to copy ALL commits from a branch to another
  - **We can cherry pick those that are of interest**
- `git cherry-pick <SHA1> <SHA2> <SHA3>`



# DEALING WITH THE REMOTE REPO



**github**  
SOCIAL CODING

# CREATE A REPO IN GITHUB

Explore



New repository

Import repository

New gist

New organization

Customize your pinned repositories

Repositories 32

Following

ProjetoIntegrador

## Create a new repository

A repository contains all the files for your project, including the revision history.

Owner



igorsteinmacher ▾

Repository name

MyRepo ✓

Great repository names are short and memorable. Need inspiration? How about [try something new](#)

Description (optional)

This is a demo project!



**Public**

Anyone can see this repository. You choose who can commit.



**Private**

You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're creating a new repository.

Add .gitignore: **None** ▾

Add a license: **None** ▾



Create repository

# CLONING TO A LOCAL REPO

- That's simple! Cloning means bringing all the history to a local repo
  - `git clone https://github.com/<owner>/<repo>`
- Testing it out
  - `git clone https://github.com/NAU-OSS/githandson.git`

*Cloning into 'githandson'...*

*warning: You appear to have cloned an empty repository.*

# WORKING IN THIS REPO

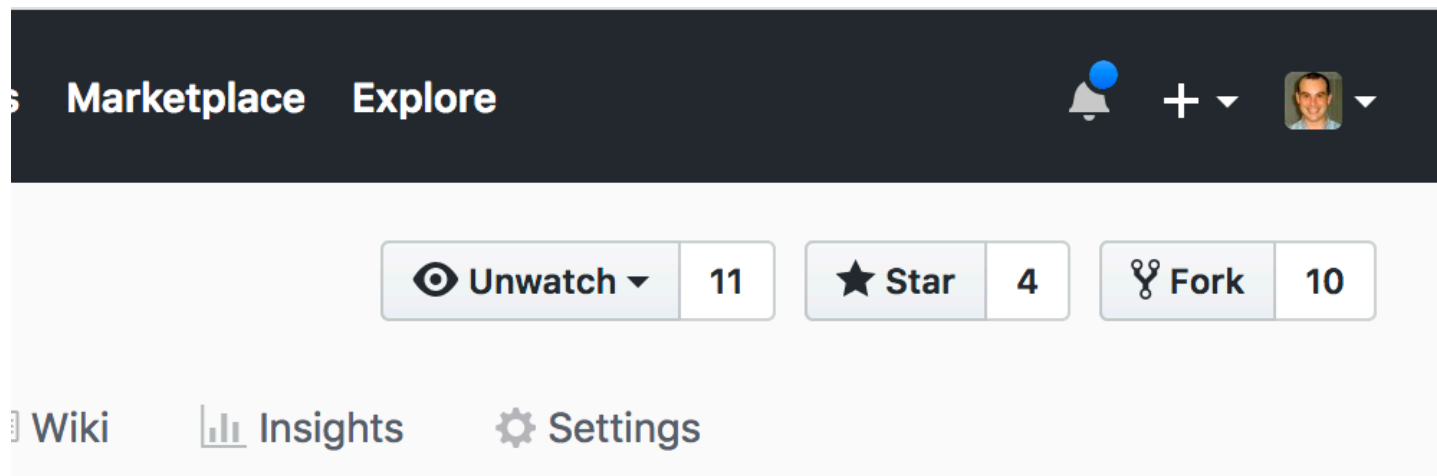
- Some different commands to deal with remote repo
  - **git branch -r**
  - **git pull** //pulls everything from the remote repo and updates the local repo
  - **git fetch** //pulls changes from remote repos, but it doesn't integrate any of this new data into your working files
  - **git push**
    - **git push <remoteName> <branchName>** //push your local changes to an online repository (git push **origin master**)
    - **git push <remoteName> <localBranchName>:<remoteBranchName>**  
// This pushes the LOCALBRANCHNAME to your REMOTENAME, but it is renamed to REMOTEBRANCHNAME.

# USUAL WORKFLOW

- git clone
  - branch out to add your changes locally
  - your adds/commits
  - pull changes to your local repo
  - merge your branch back (LOCALLY)
    - Resolve any conflict
  - push changes back to the remote repo

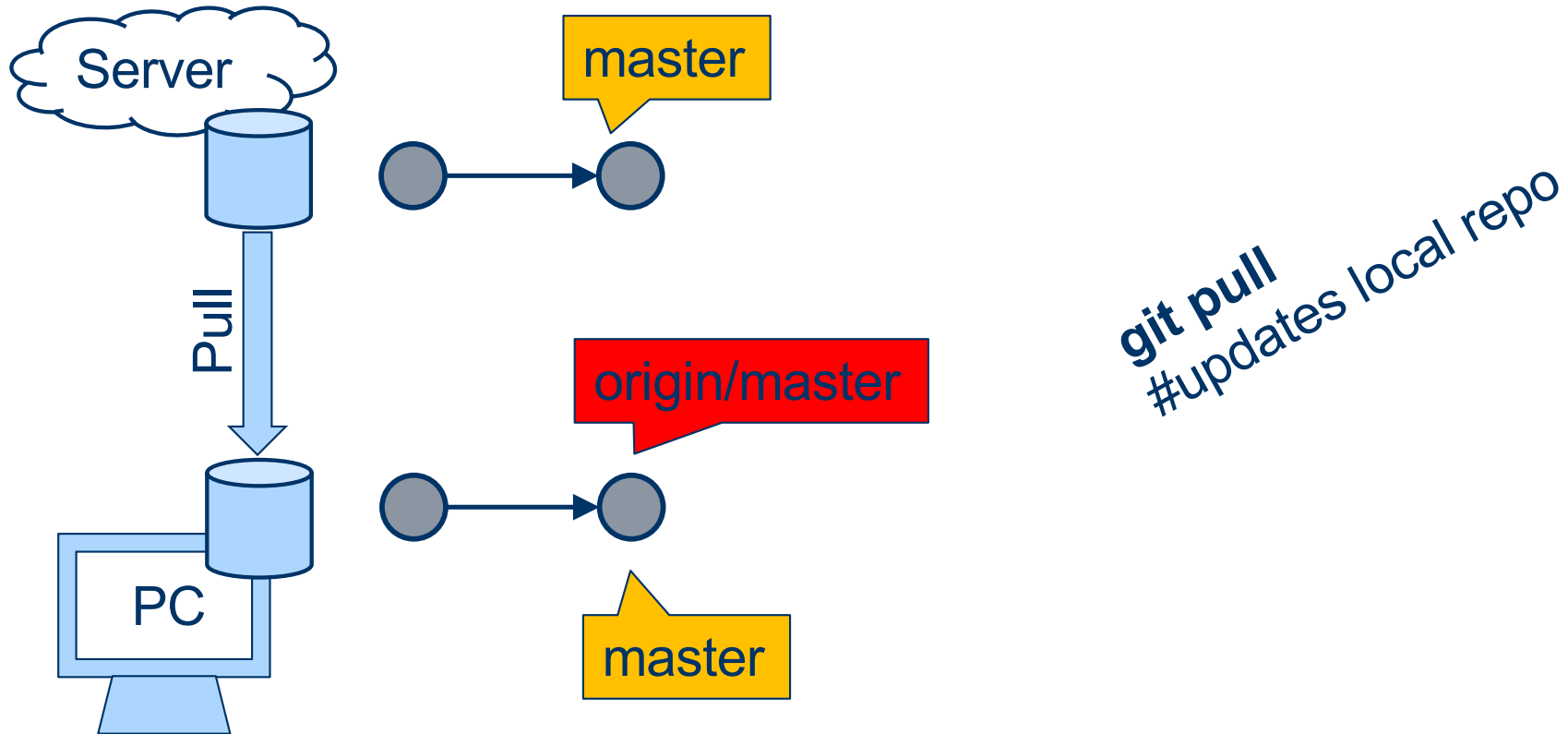
# GITHUB WORKFLOW

- Fork + pull-request
  - You usually create a fork for your repo
    - “A **fork** is a copy of a repository. **Forking** a repository allows you to freely experiment with changes without affecting the original project”
  - Creating a fork



- Then, you usually clone your fork... work, and send a pull request against the main repo

# EXAMPLE

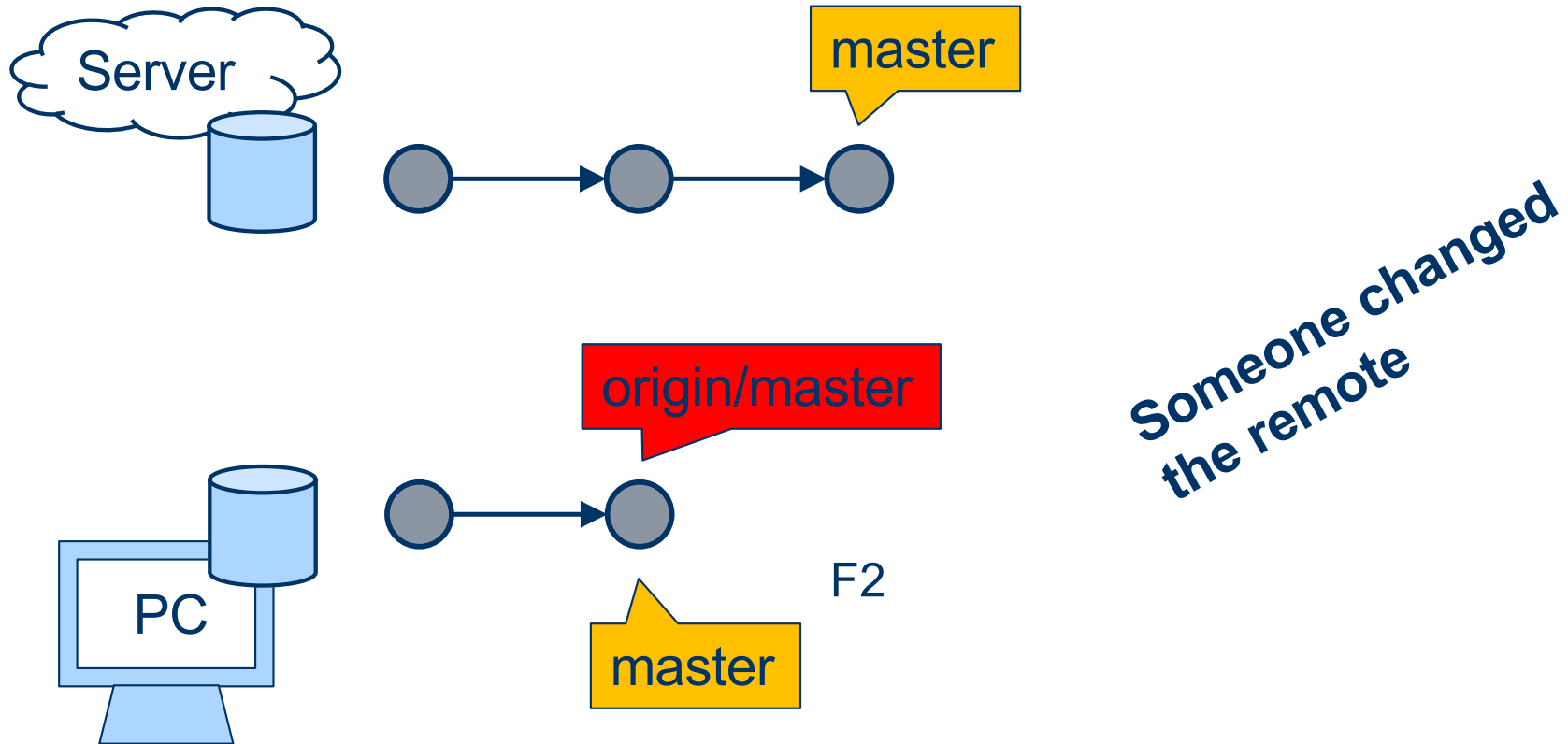


**master** is a local branch

**origin/master** is a remote branch (a *local copy* of the branch named "master" on the remote named "origin")

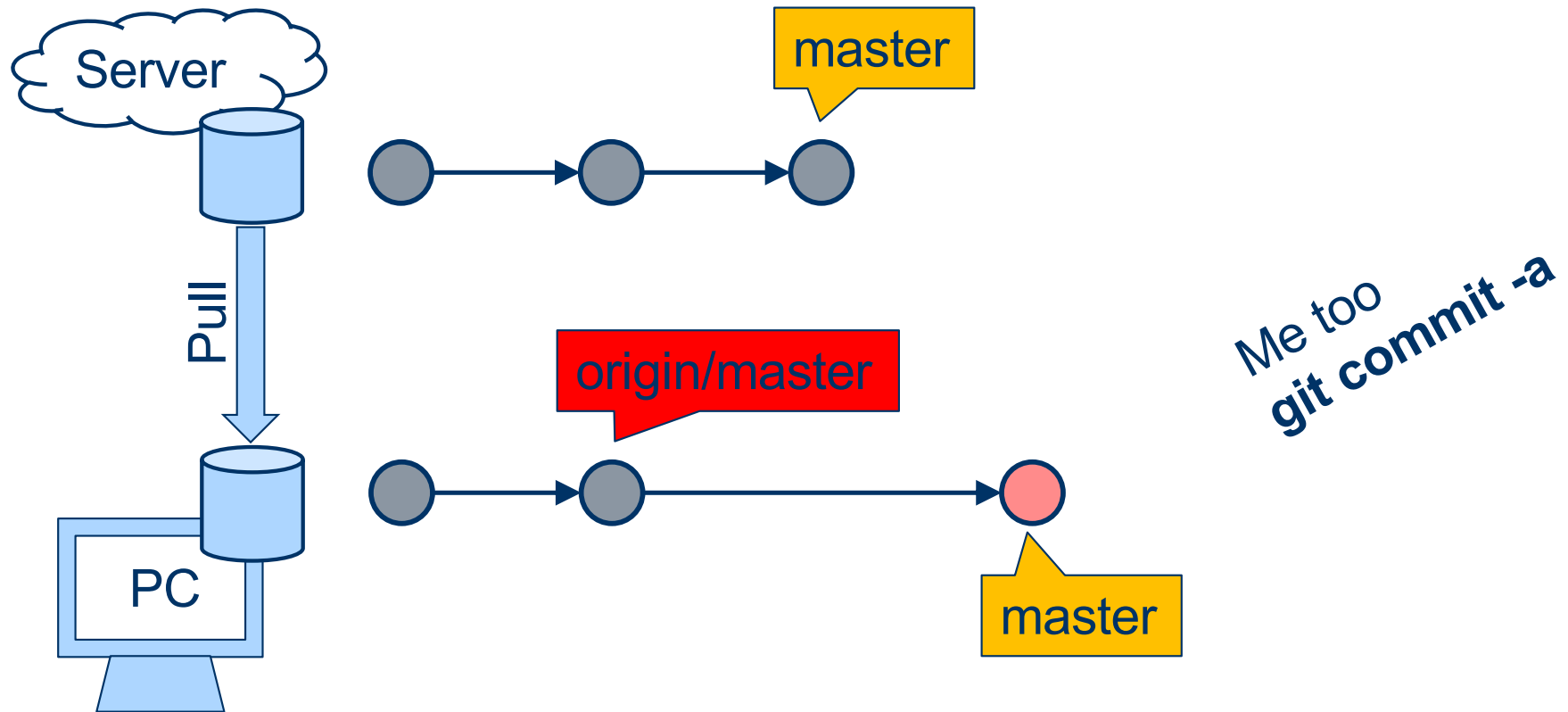
**origin** is a remote

# EXAMPLE

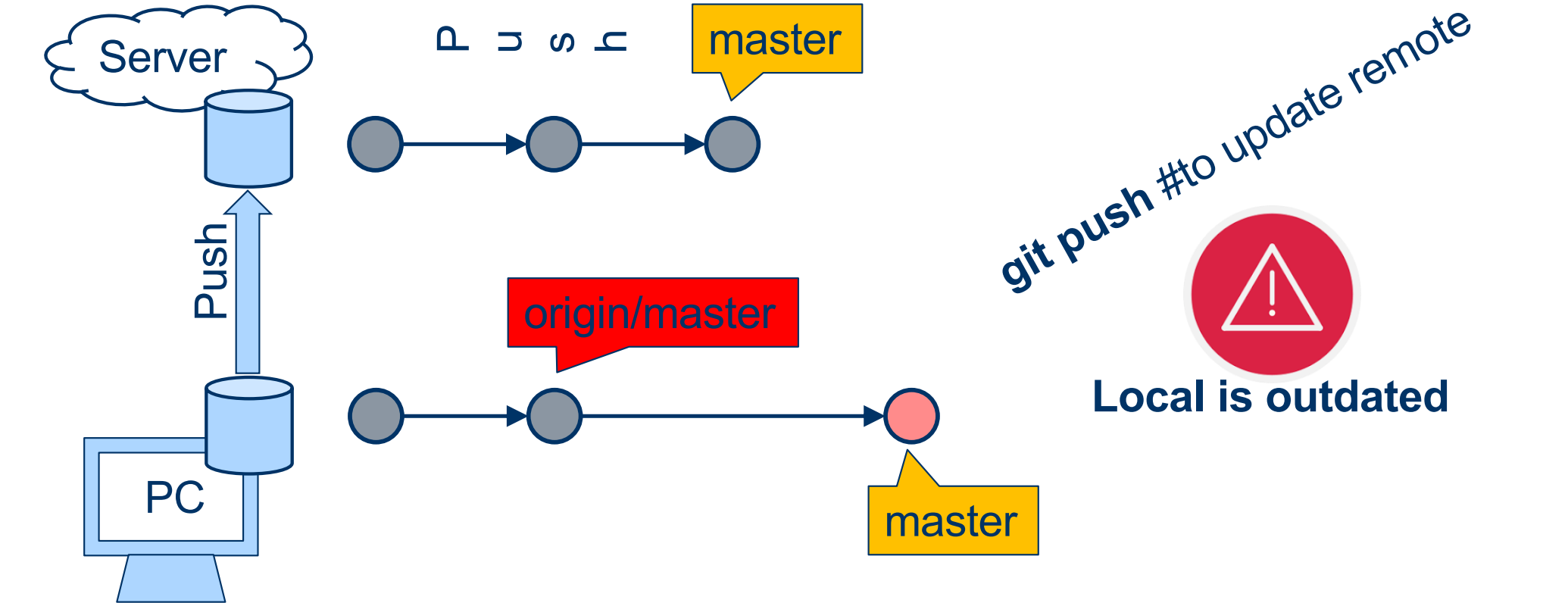




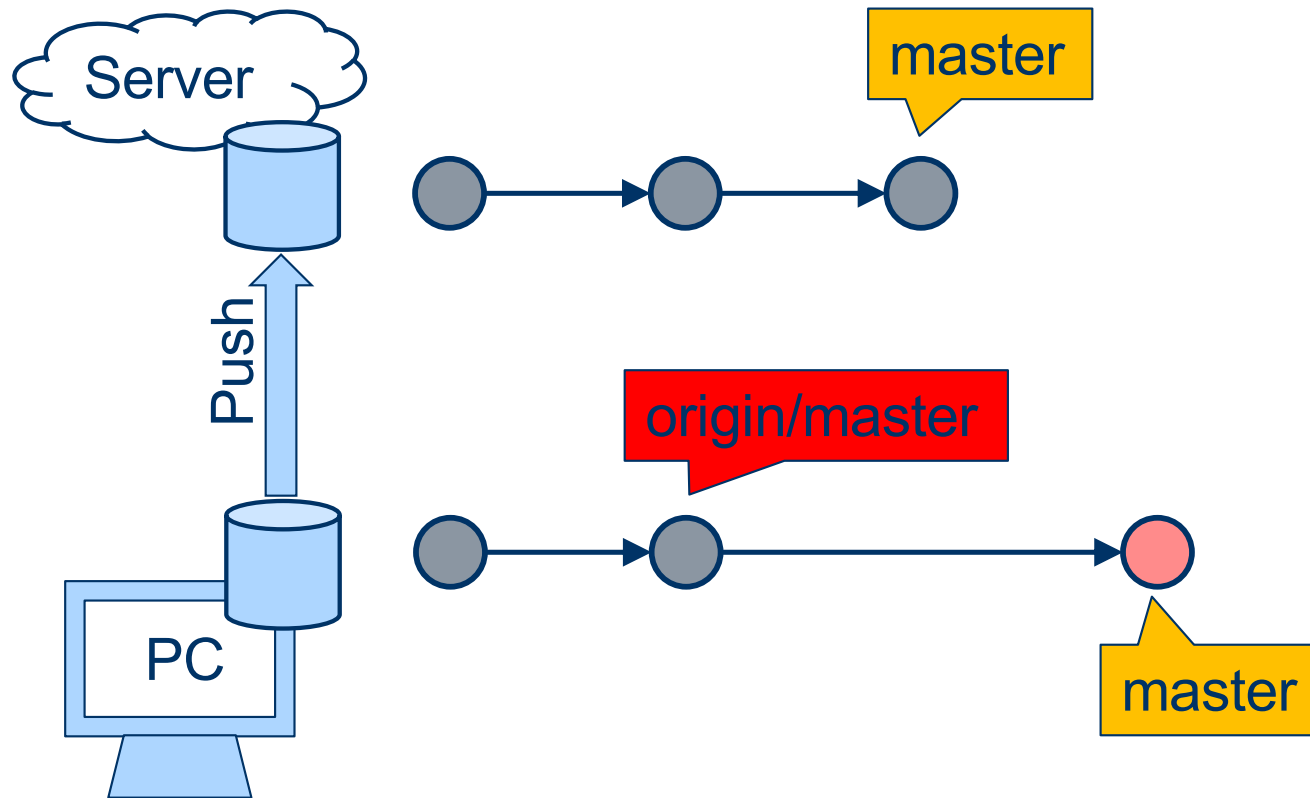
# EXAMPLE



## EXAMPLE



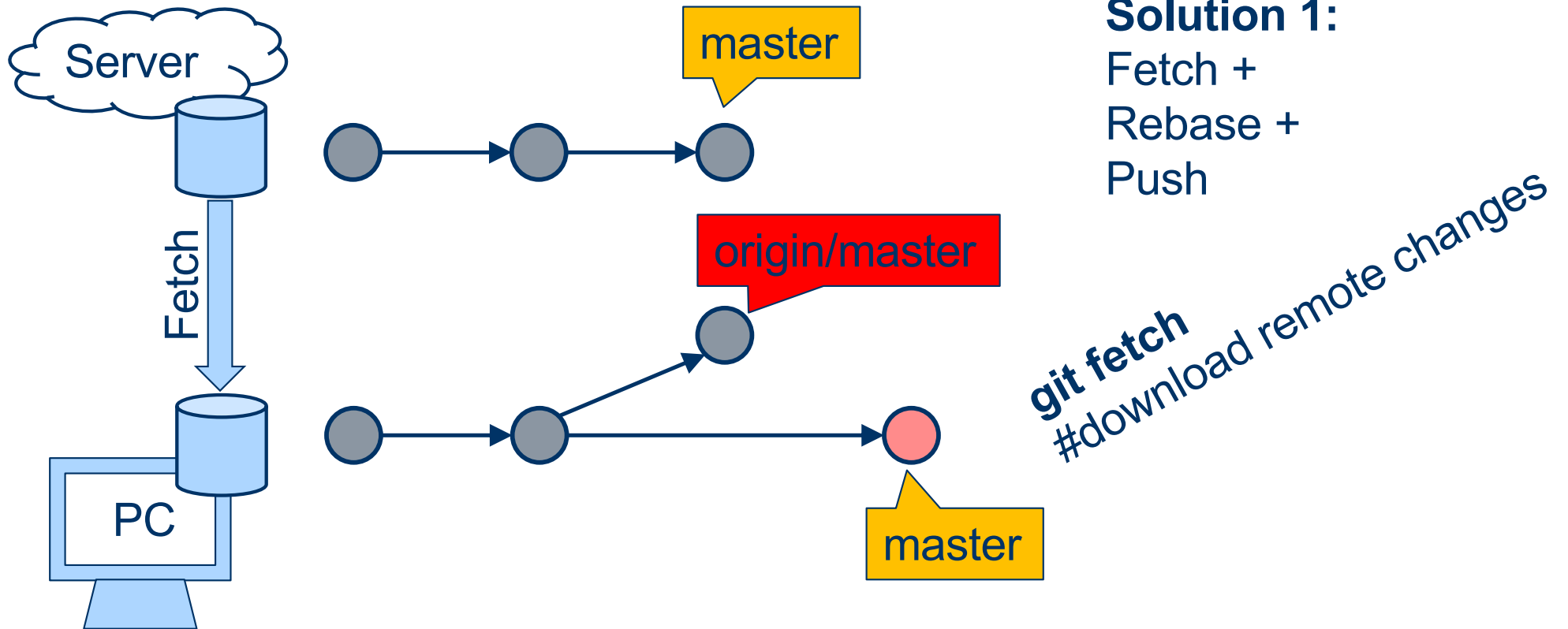
# EXAMPLE



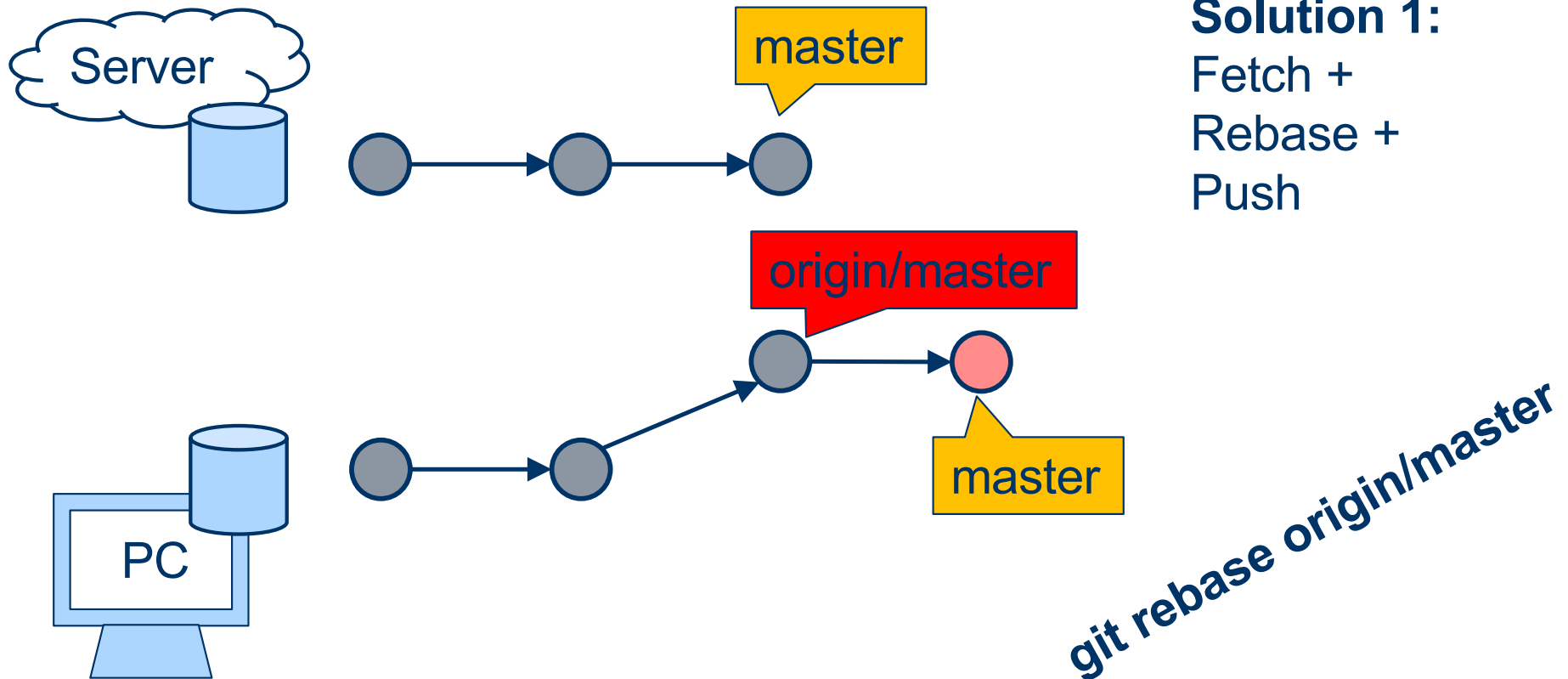
## Solution 1:

Fetch +  
Rebase +  
Push

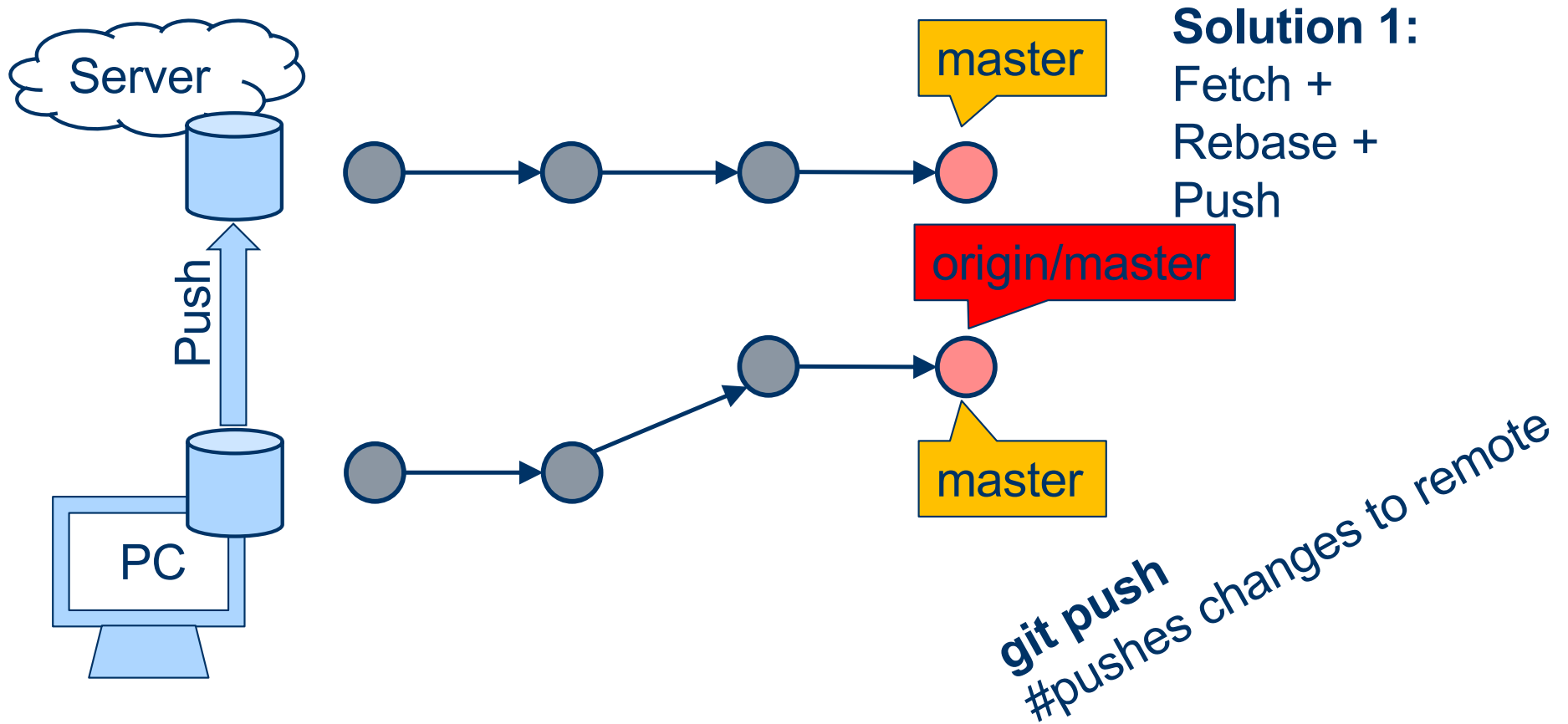
# EXAMPLE



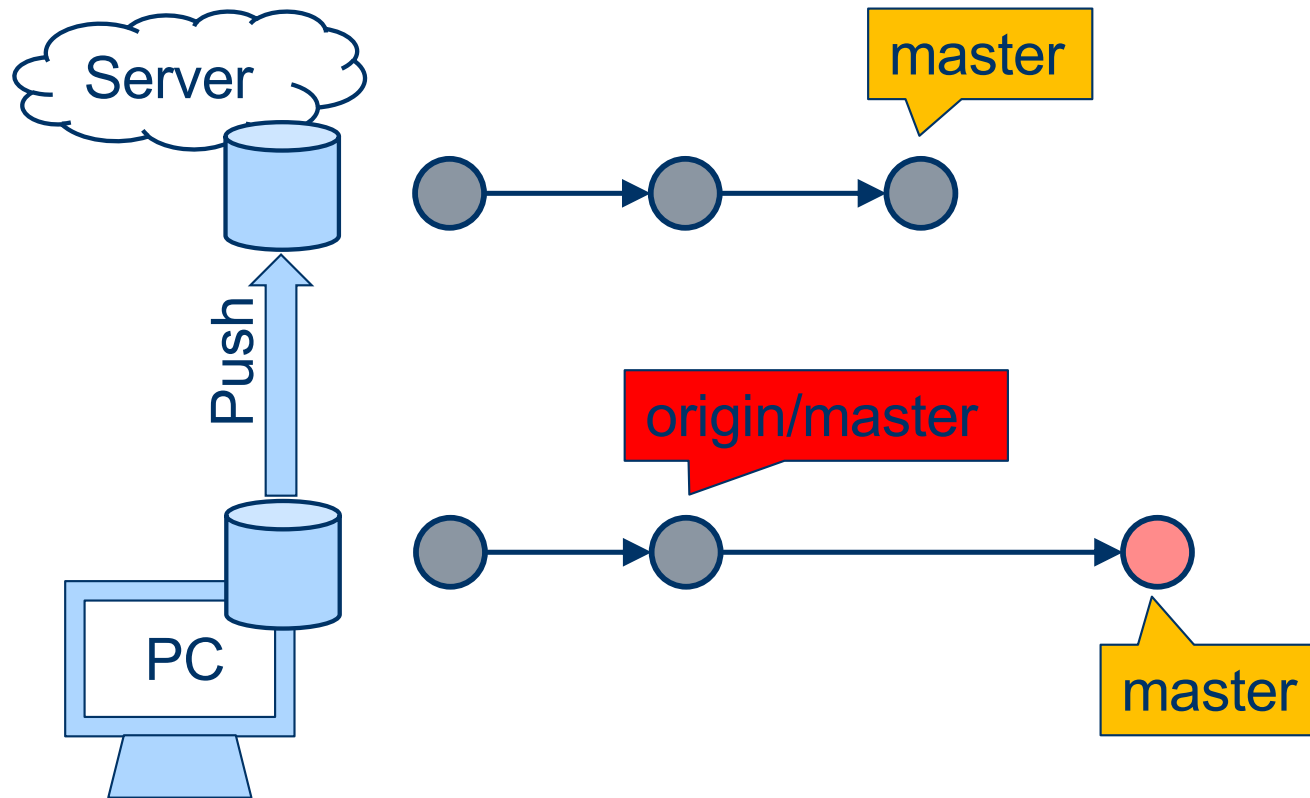
# EXAMPLE



# EXAMPLE

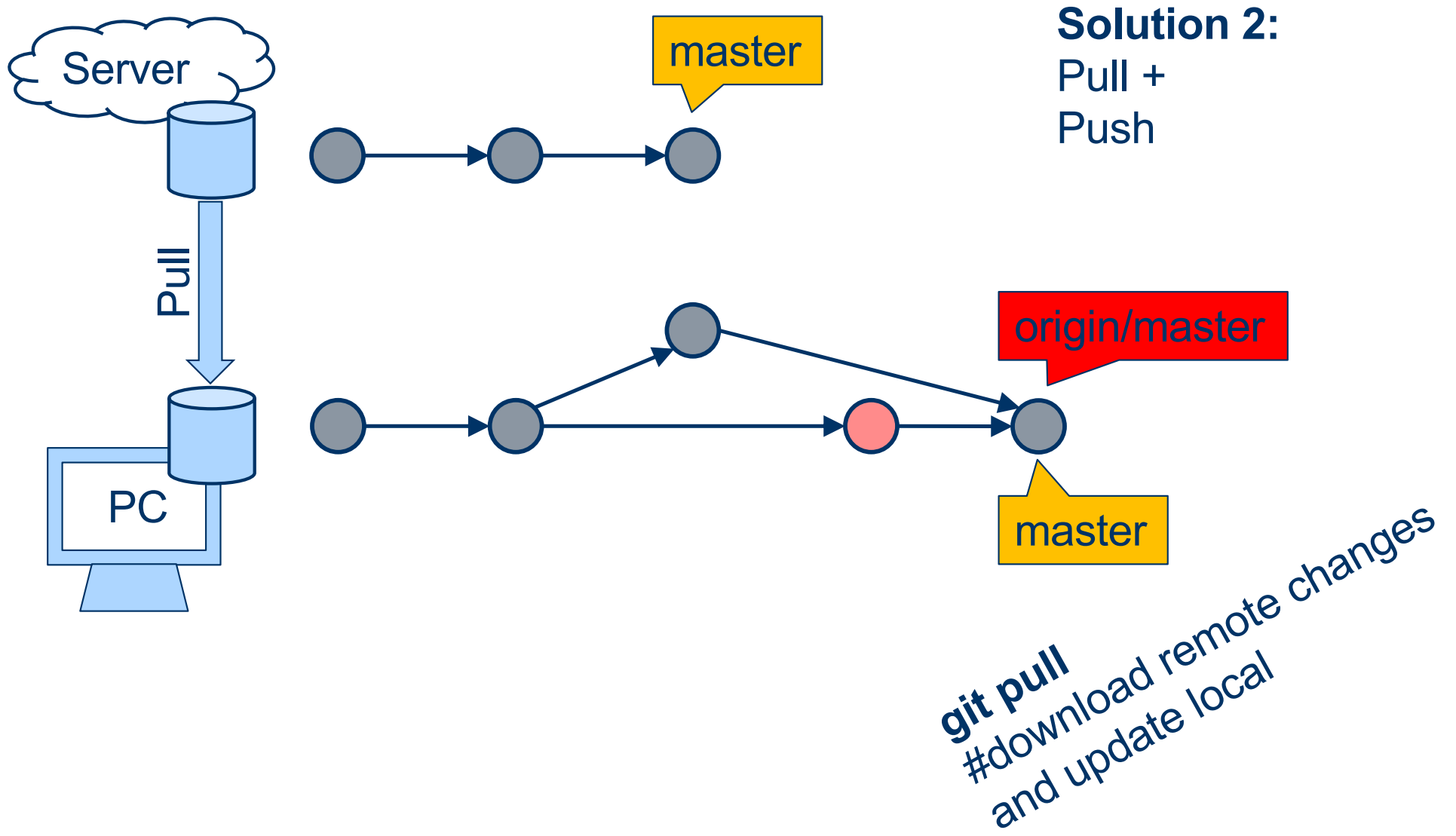


# EXAMPLE



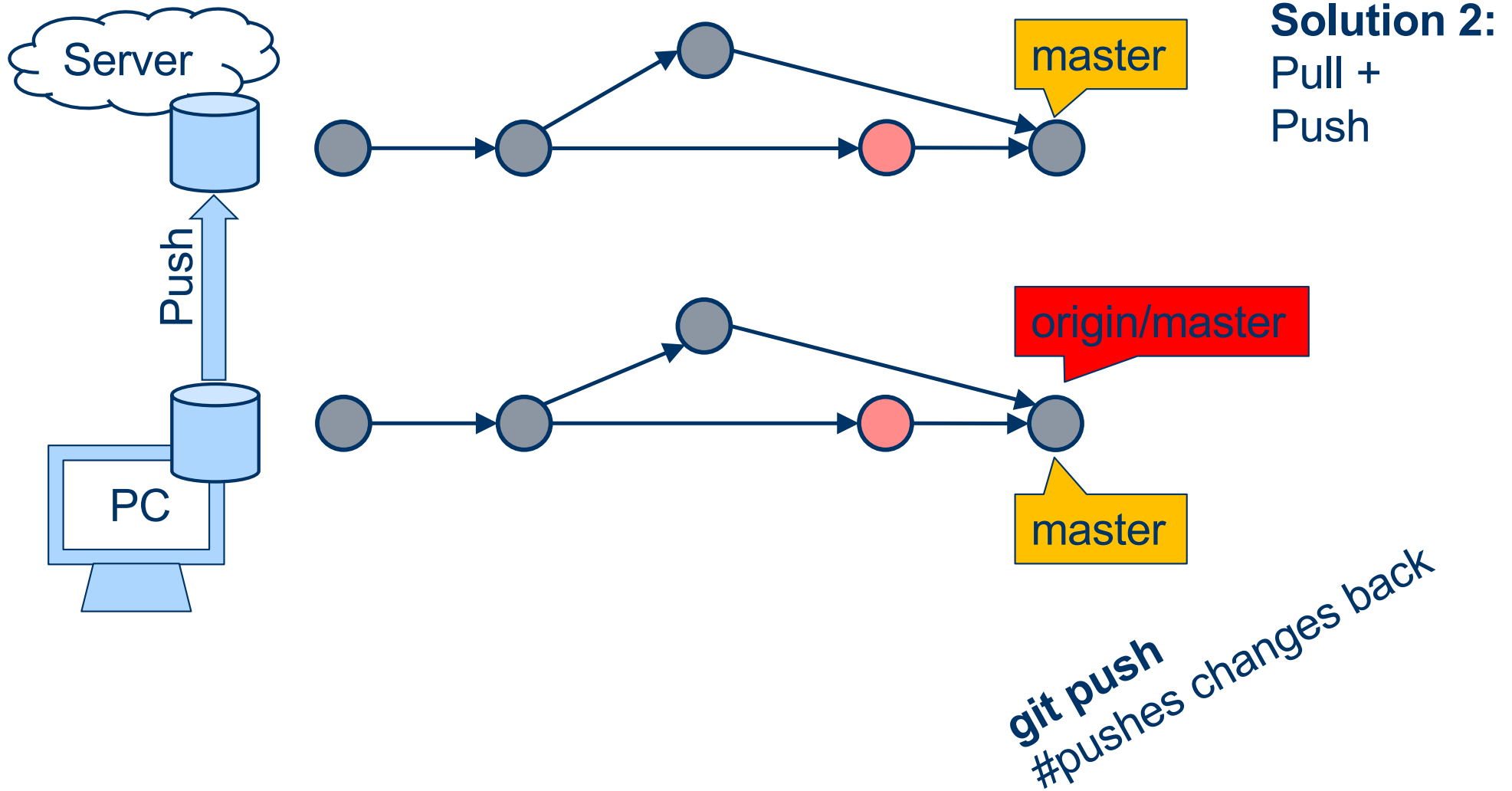
**Solution 2:**  
Pull +  
Push

# EXAMPLE





# EXAMPLE



# A PULL REQUEST EXAMPLE

- (GitHub) Fork
- (CLI) Clone (the fork)
- (CLI) Commits
- (CLI) Push
- (GitHub) Send Pull Request
  - **Follow it**
  - **Revise it**
  - **Update it**
- Keep your fork up-to-date
  - <https://help.github.com/articles/syncing-a-fork/>