

# The Why and How of git

*"The Stupid Content Tracker"*

J. Rötter<sup>1</sup>, J. Rosenzweig<sup>1</sup>

<sup>1</sup>University of Florida

2021-04-02

# What you will learn:

- The **what, why, when, and how** of Git



# What you will learn:

- The **what, why, when, and how** of Git
- What is Git?



# What you will learn:

- The **what, why, when, and how** of Git
- What is Git?
  - ▶ According to Cambridge English Dictionary:



**git**

**noun** [ C ] UK informal

UK  /git/ US  /git/

---

+ ⋮

**a person, especially a man, who is stupid or unpleasant:**

- *You stupid/lying git!*
- *He's a miserable old git.*

# What you will learn:

- The **what, why, when, and how** of Git
- What is Git?
  - ▶ According to Cambridge English Dictionary:
  - ▶ Version control software.



# git

git

**noun** [ C ] UK informal

UK  /git/ US  /git/

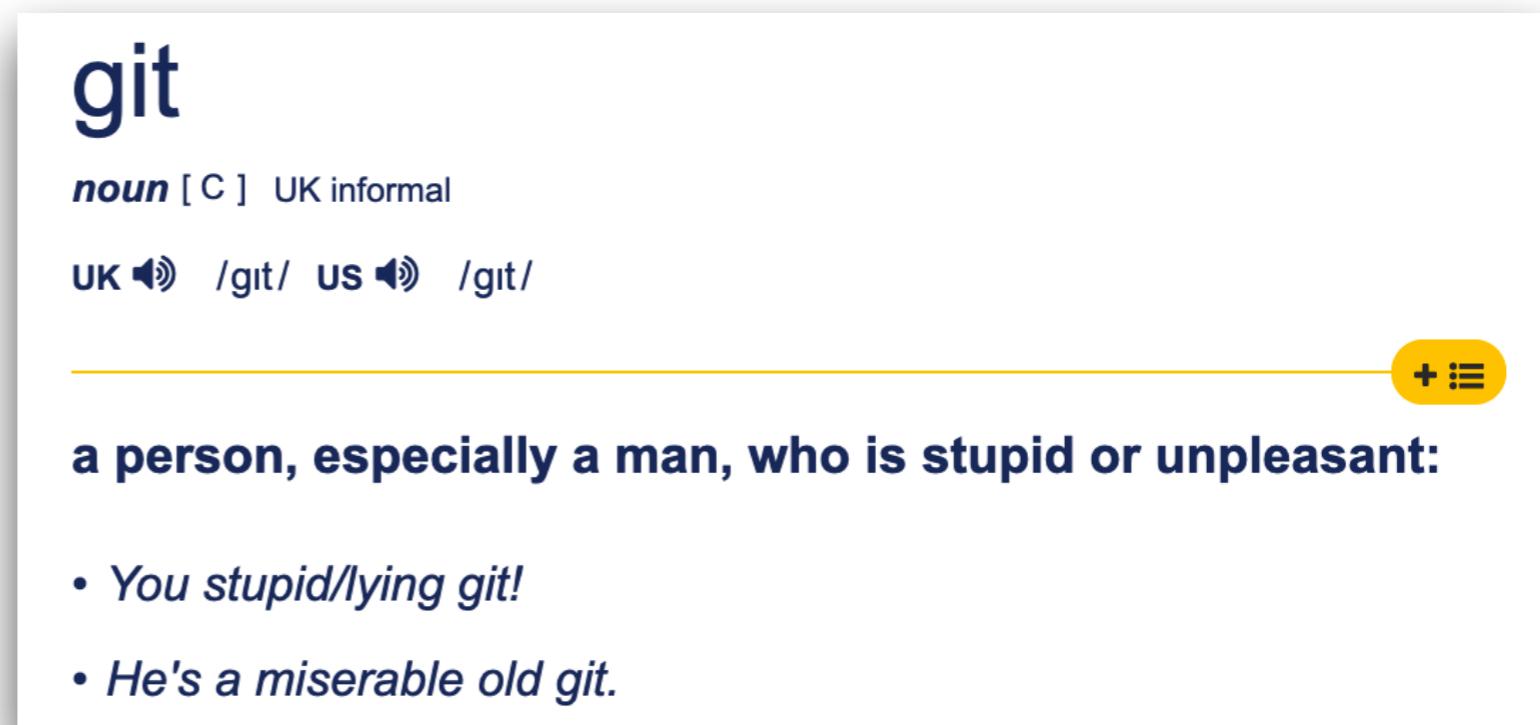
+ ⋮

**a person, especially a man, who is stupid or unpleasant:**

- *You stupid/lying git!*
- *He's a miserable old git.*

# What you will learn:

- The **what, why, when, and how** of Git
- What is Git?
  - ▶ According to Cambridge English Dictionary:
  - ▶ Version control software.
- Why should you use Git?



git

**noun [ C ]** UK informal

UK  /git/ US  /git/

---

+ ⋮

**a person, especially a man, who is stupid or unpleasant:**

- *You stupid/lying git!*
- *He's a miserable old git.*

# What you will learn:

- The **what, why, when, and how** of Git
- *What is Git?*
  - ▶ According to Cambridge English Dictionary:
  - ▶ Version control software.
- *Why should you use Git?*
  - ▶ Save code
  - ▶ Share code.
  - ▶ Modify **other people's** code.
  - ▶ Revert back to previous (working) code!
  - ▶ Public code is your **code "CV"**.
  - ▶ (You can use Git with private code)



**git**

**noun** [ C ] UK informal

UK /git/ US /git/

---

**a person, especially a man, who is stupid or unpleasant:**

- *You stupid/lying git!*
- *He's a miserable old git.*

# What you will learn:

- The **what, why, when, and how** of Git

- *What is Git?*

- According to Cambridge English Dictionary:
- Version control software.

- *Why should you use Git?*

- **Save** code
- **Share** code.
- Modify **other people's** code.
- **Revert** back to previous (working) code!
- Public code is your **code "CV"**.
- (You can use Git with private code)

- *When should you use Git?*



**git**

**noun** [ C ] UK informal

UK /git/ US /git/

---

+

**a person, especially a man, who is stupid or unpleasant:**

- *You stupid/lying git!*
- *He's a miserable old git.*

# What you will learn:

- The **what, why, when, and how** of Git
- *What is Git?*
  - ▶ According to Cambridge English Dictionary:
  - ▶ Version control software.
- *Why should you use Git?*
  - ▶ Save code
  - ▶ Share code.
  - ▶ Modify **other people's** code.
  - ▶ Revert back to previous (working) code!
  - ▶ Public code is your **code "CV"**.
  - ▶ (You can use Git with private code)
- *When should you use Git?*
  - ▶ Every time you code.



**git**

**noun** [ C ] UK informal

UK /git/ US /git/

---

**a person, especially a man, who is stupid or unpleasant:**

- *You stupid/lying git!*
- *He's a miserable old git.*

# What you will learn:

- The **what, why, when, and how** of Git

- *What is Git?*

- According to Cambridge English Dictionary:
- Version control software.



- *Why should you use Git?*

- **Save** code
- **Share** code.
- Modify **other people's** code.
- **Revert** back to previous (working) code!
- Public code is your **code "CV"**.
- (You can use Git with private code)

- *When should you use Git?*

- **Every time you code.**

- *How do you use Git?*

**git**

**noun** [ C ] UK informal

UK /git/ US /git/

---

**a person, especially a man, who is stupid or unpleasant:**

- *You stupid/lying git!*
- *He's a miserable old git.*

# What you will learn:

- The **what, why, when, and how** of Git

- *What is Git?*

- According to Cambridge English Dictionary:
- Version control software.



- *Why should you use Git?*

- **Save** code
- **Share** code.
- Modify **other people's** code.
- **Revert** back to previous (working) code!
- Public code is your **code "CV"**.
- (You can use Git with private code)

- *When should you use Git?*

- **Every time you code.**

- *How do you use Git?*

- *Step into my office...*

**git**

**noun** [ C ] UK informal

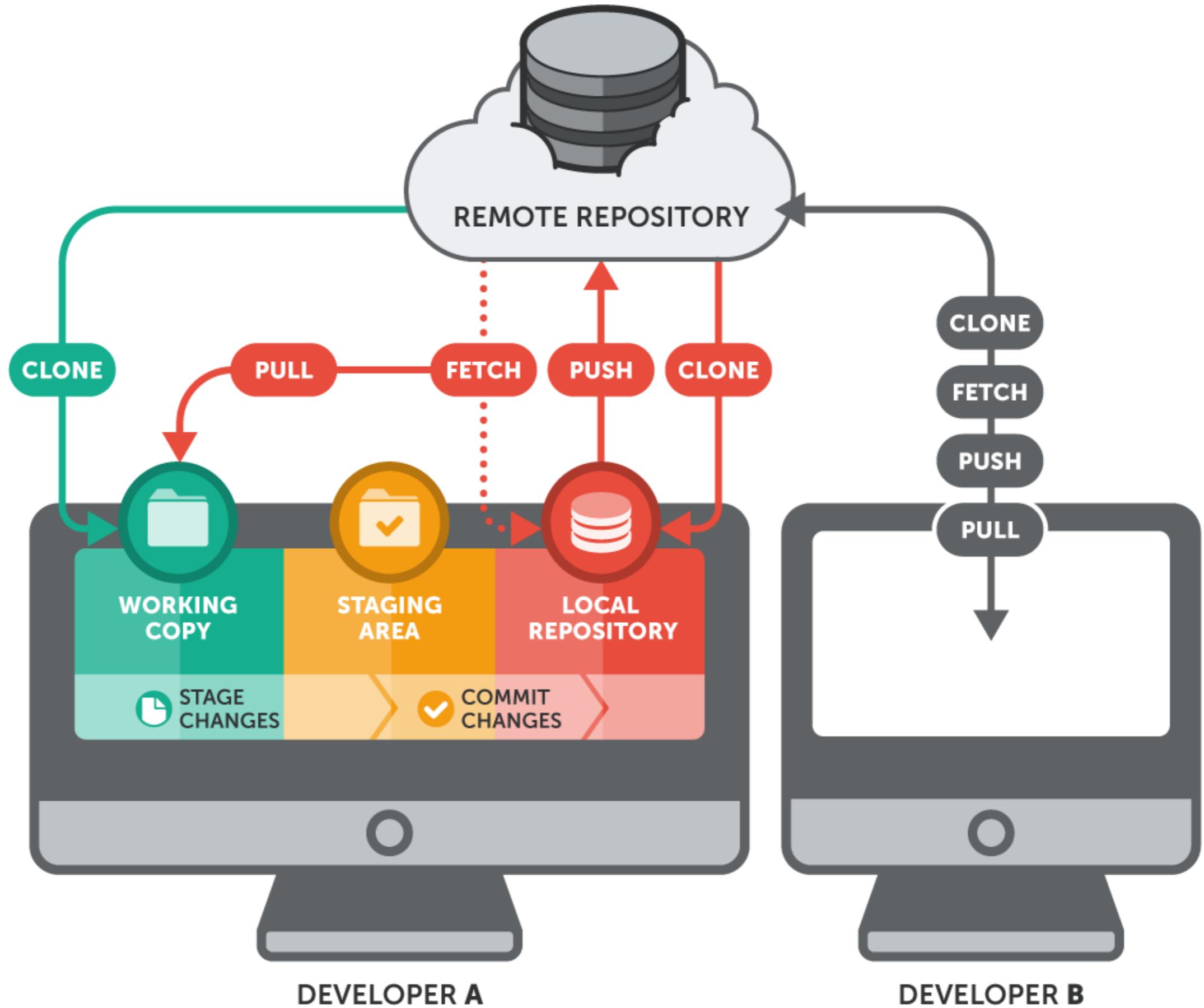
UK /git/ US /git/

---

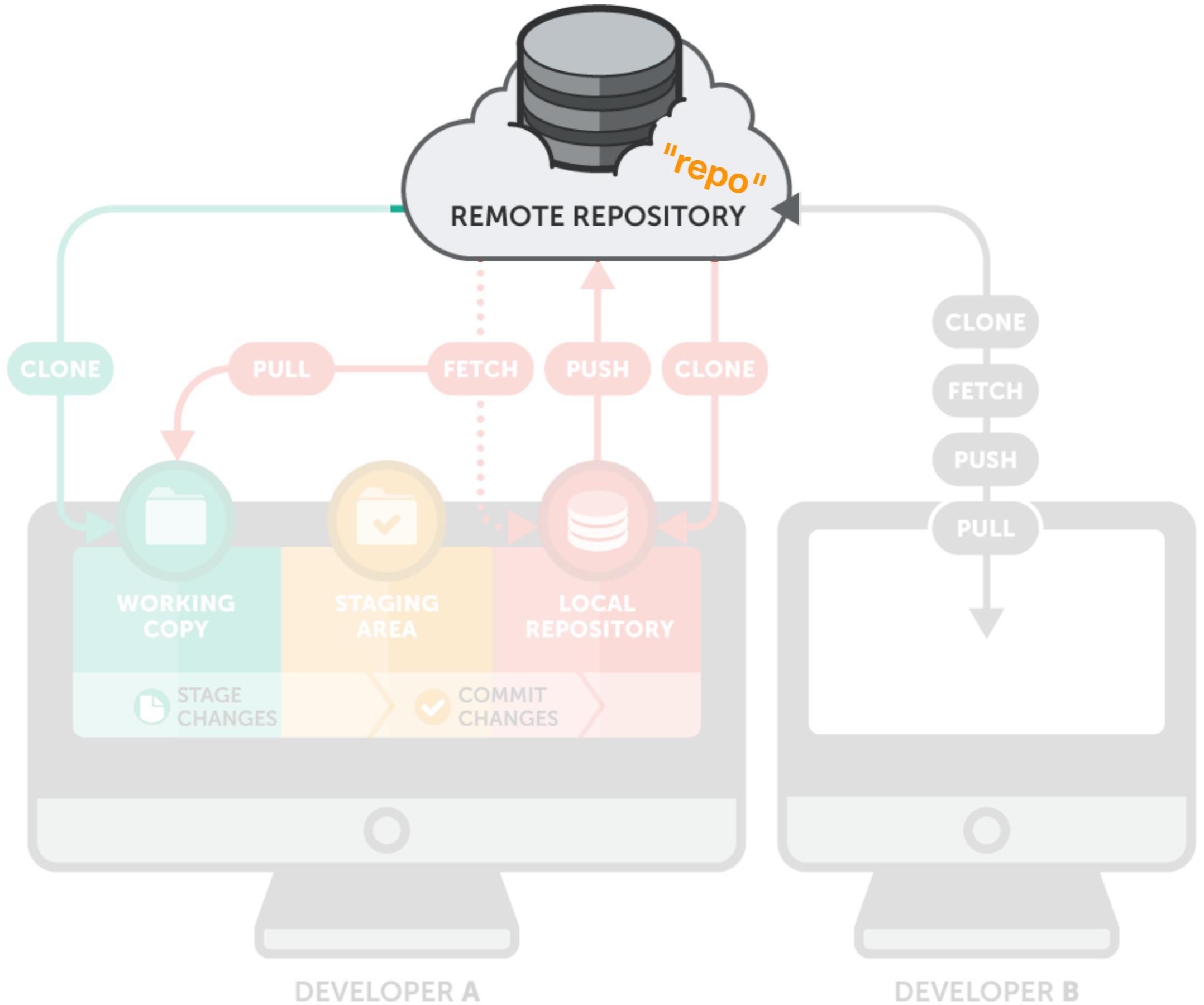
**a person, especially a man, who is stupid or unpleasant:**

- *You stupid/lying git!*
- *He's a miserable old git.*

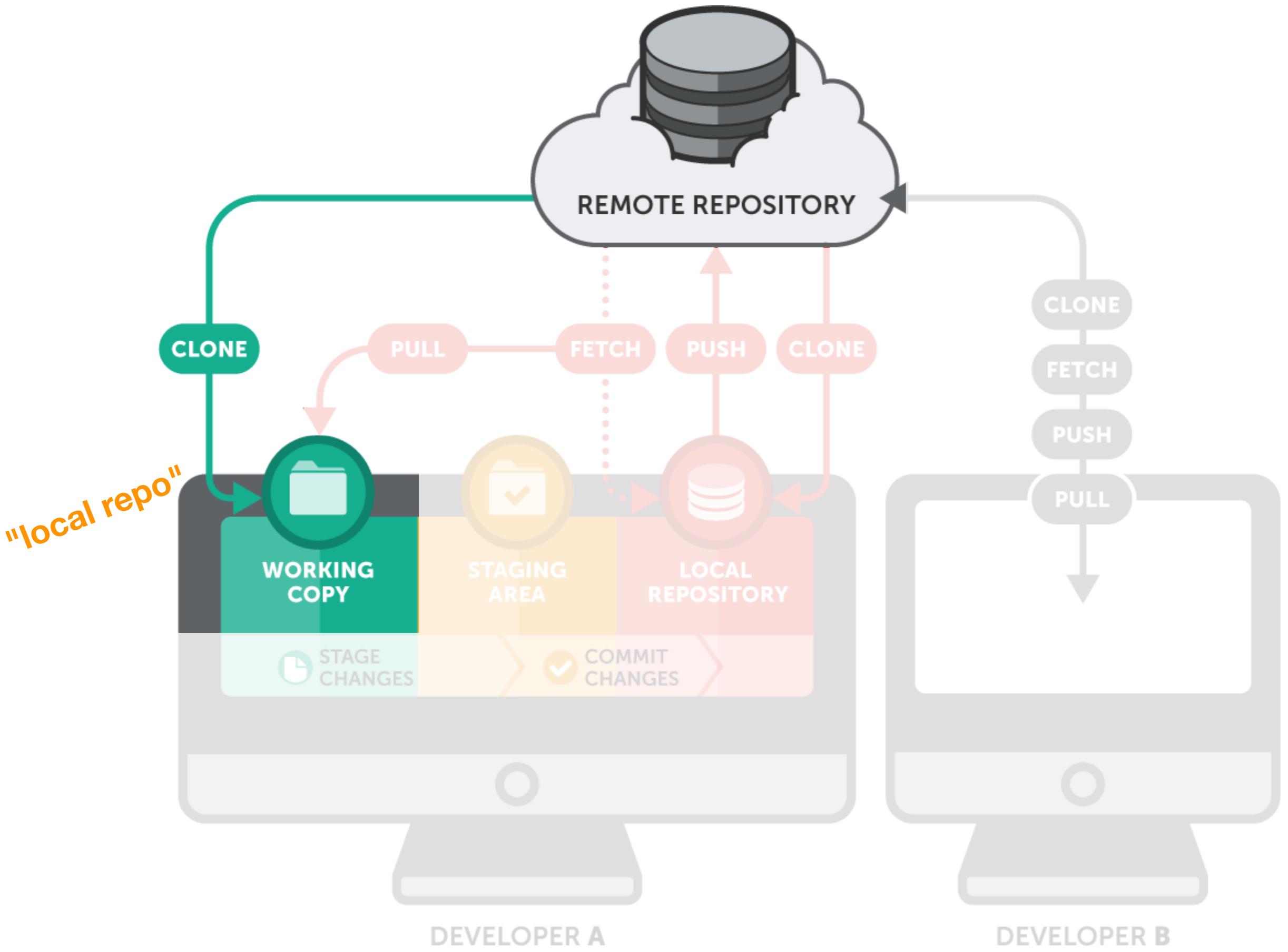
# Git the Main Idea:



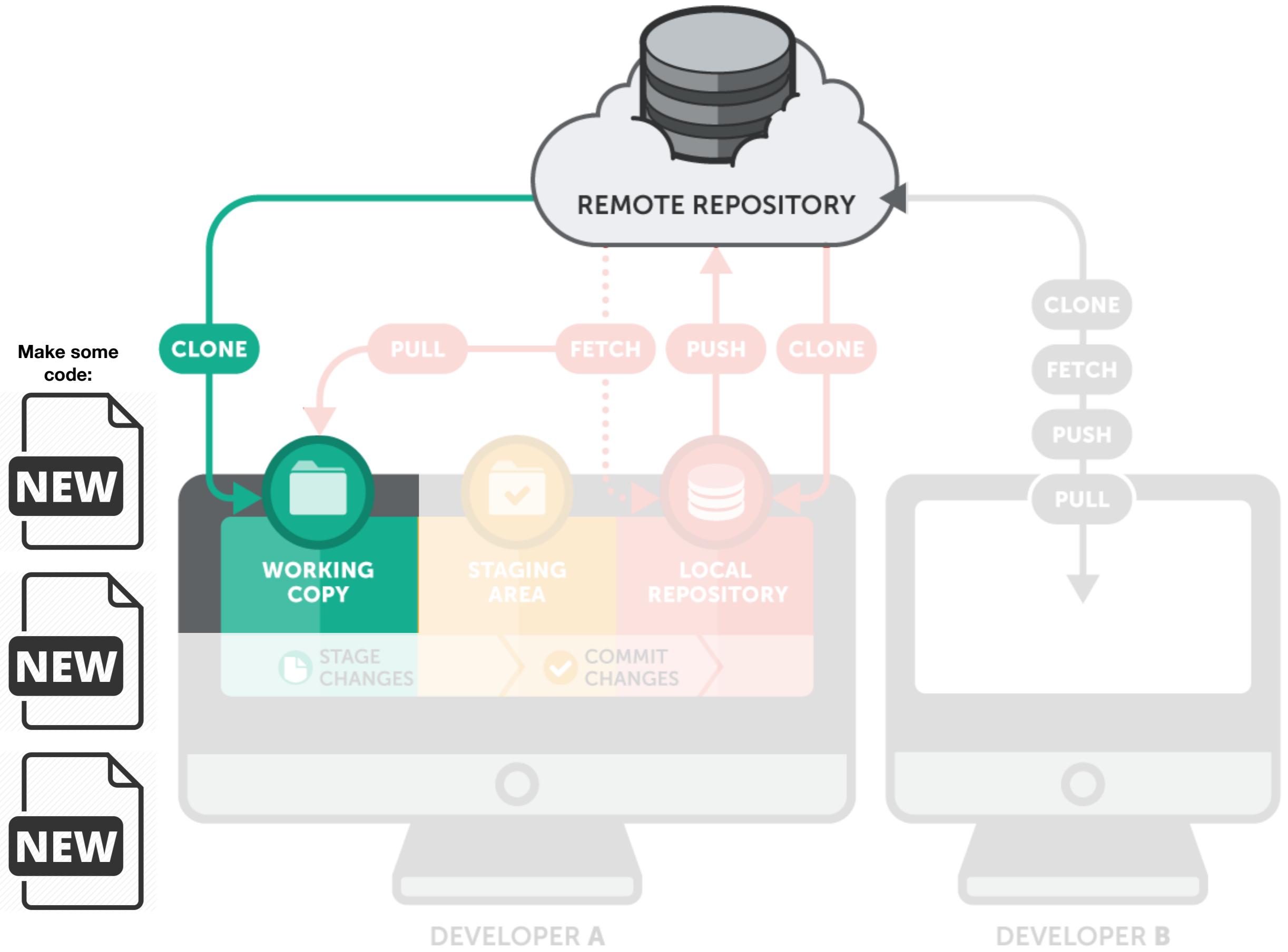
# Git the Main Idea:



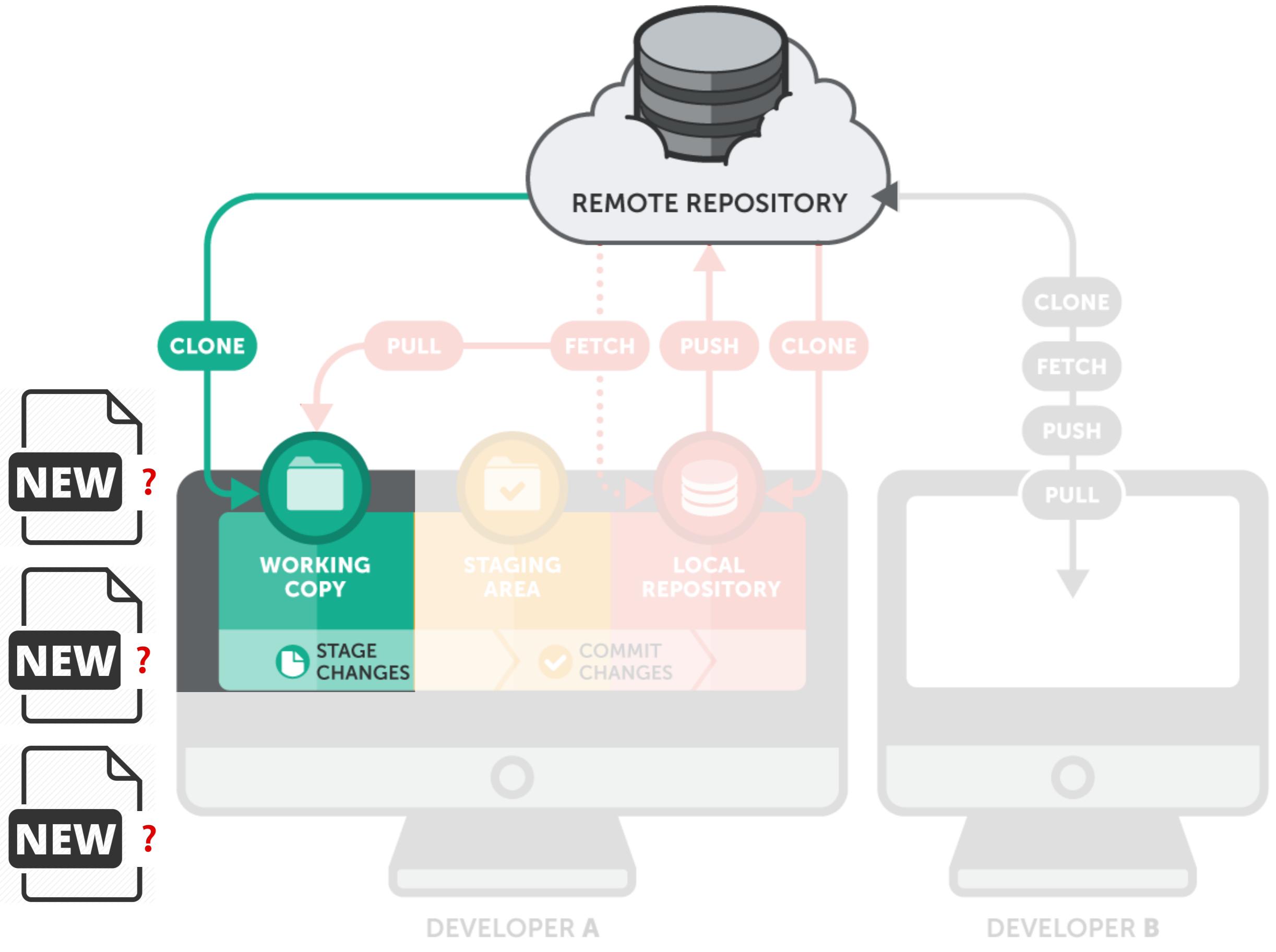
# Git the Main Idea:



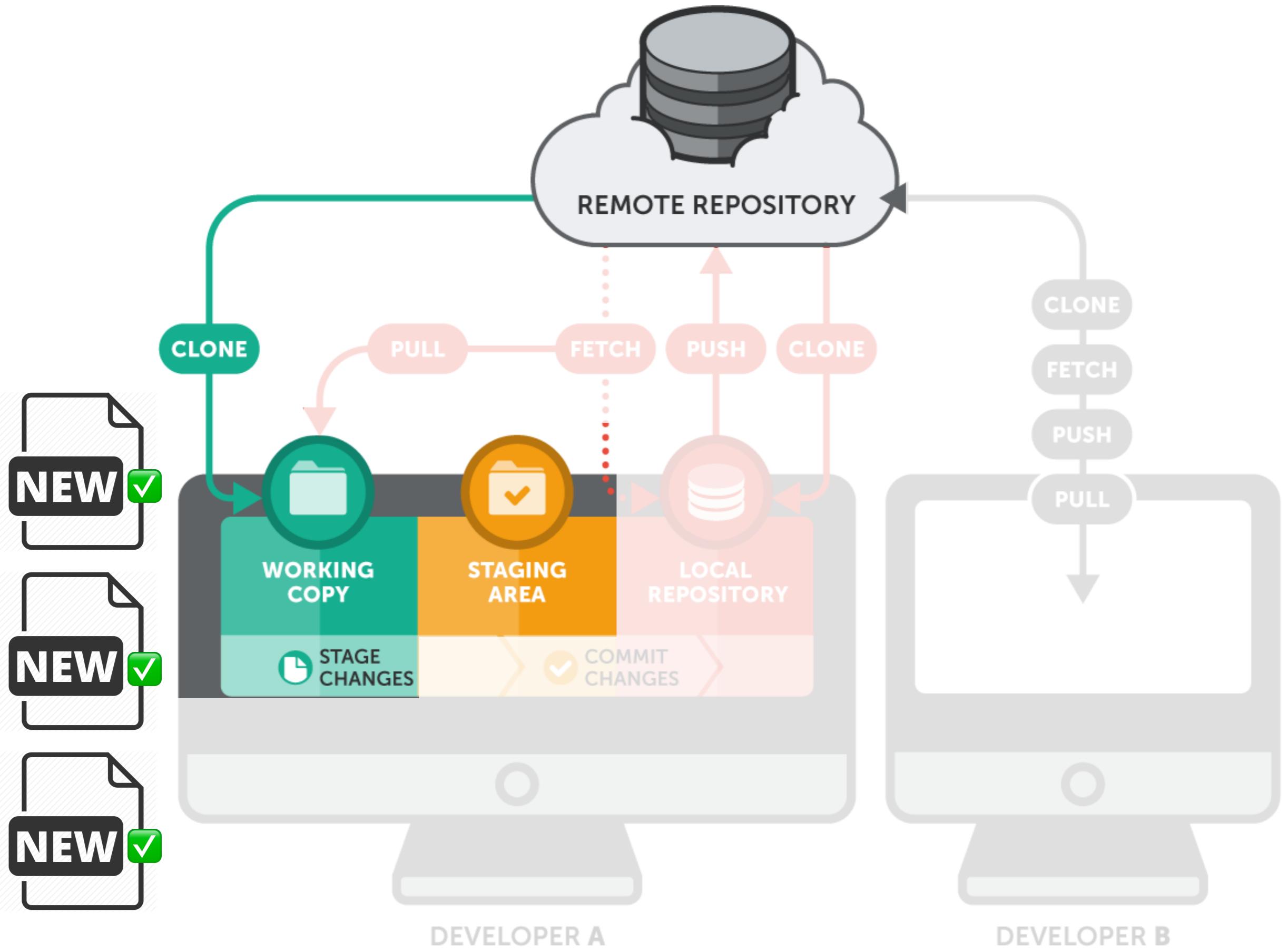
# Git the Main Idea:



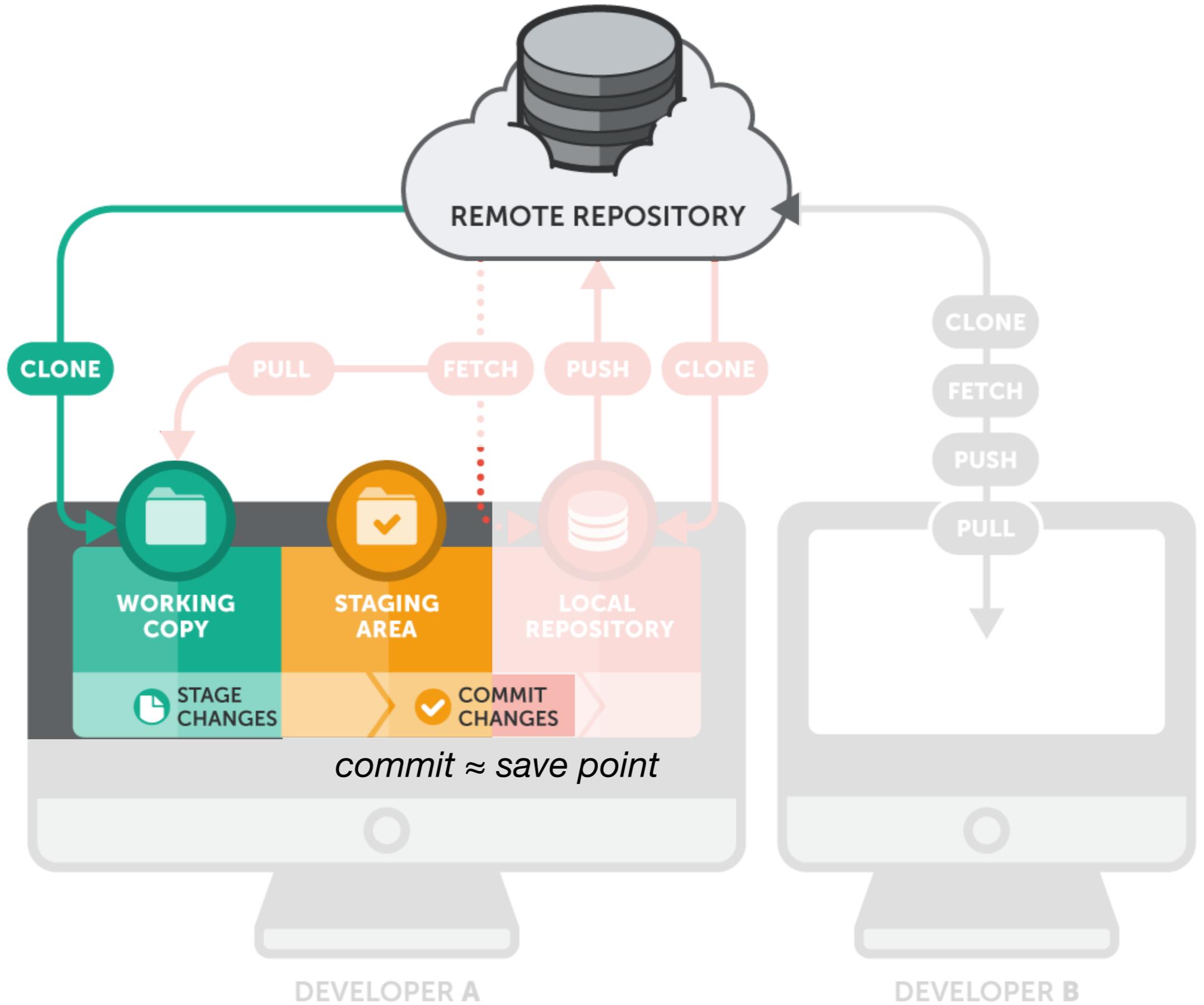
# Git the Main Idea:



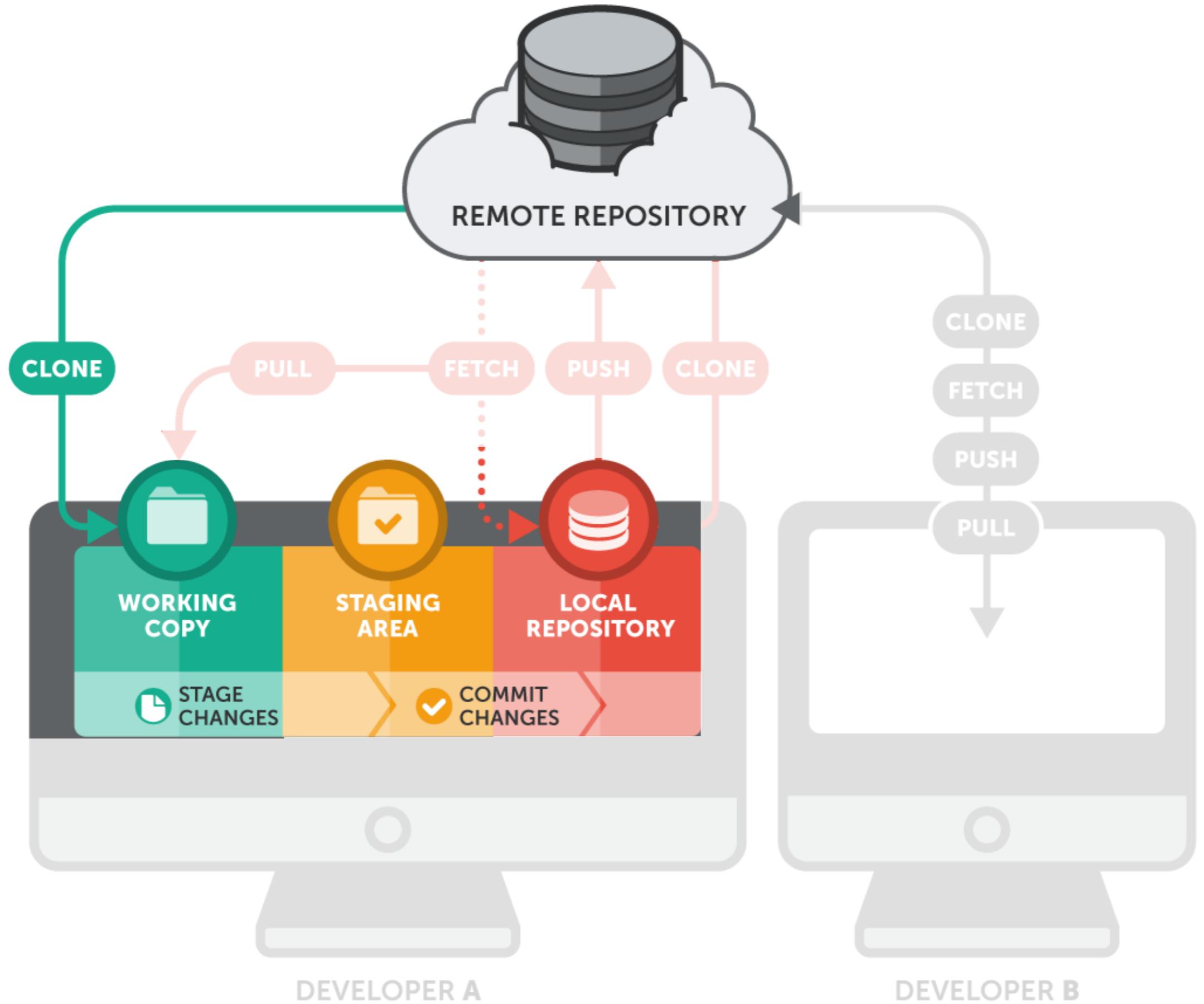
# Git the Main Idea:



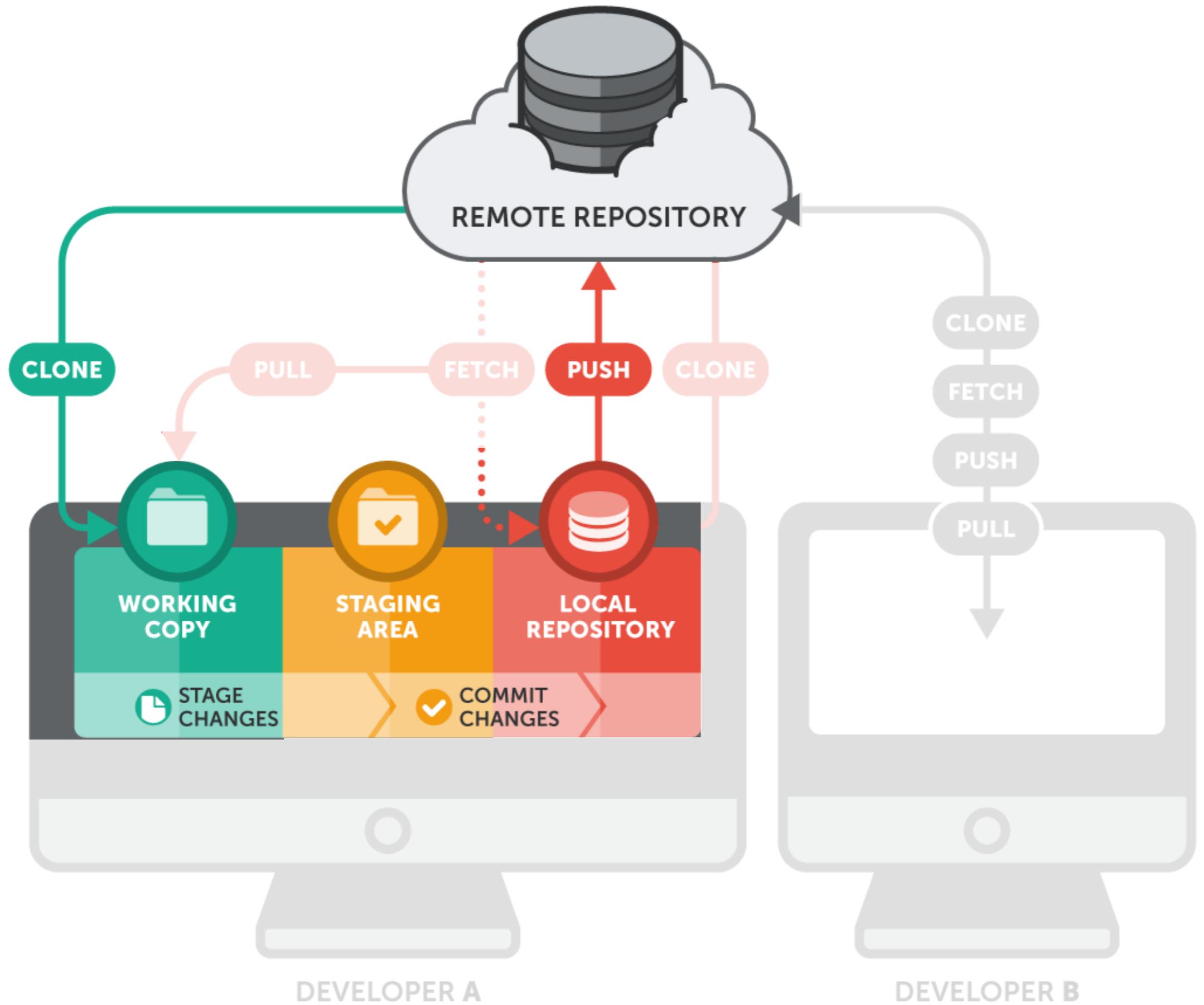
# Git the Main Idea:



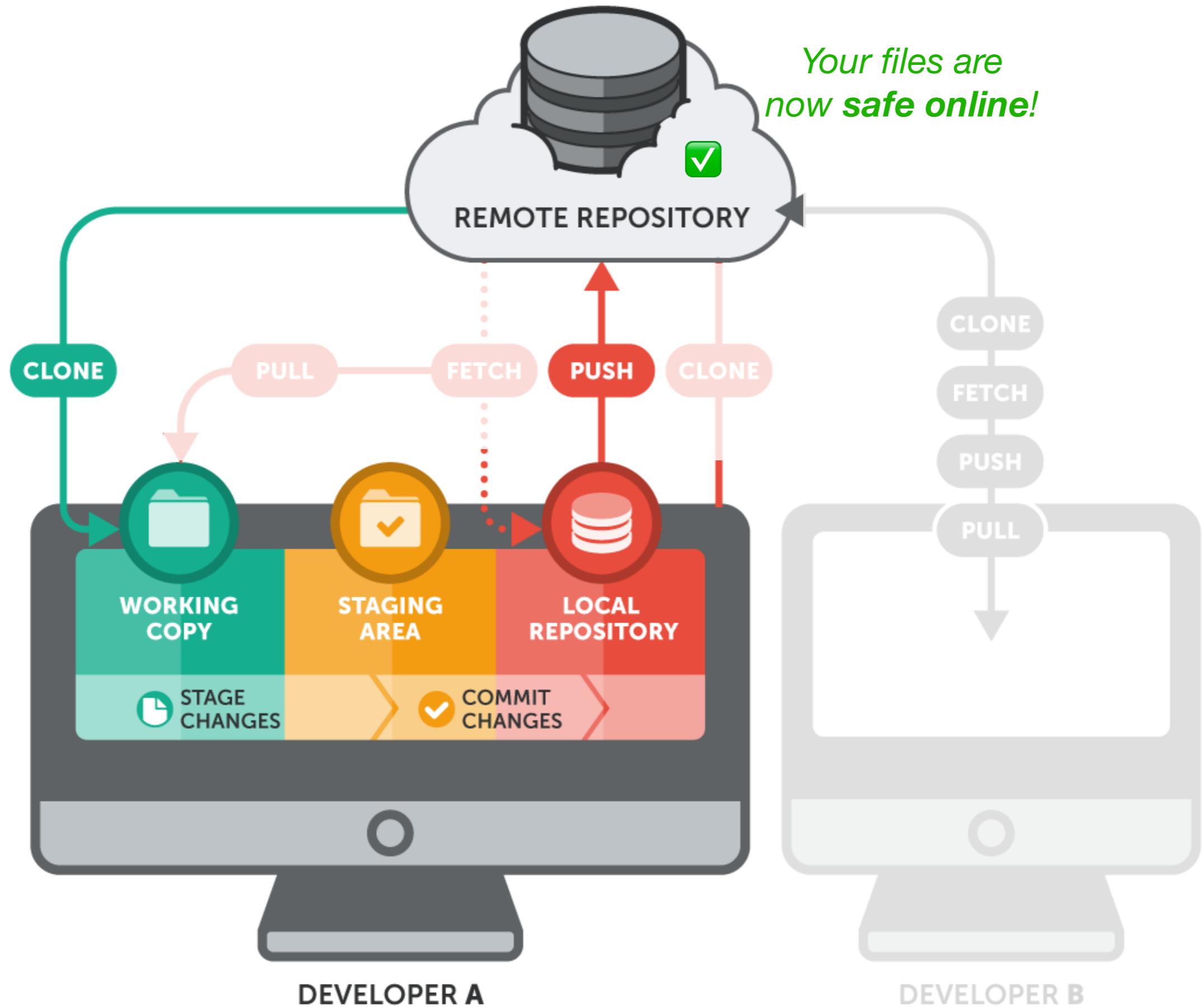
# Git the Main Idea:



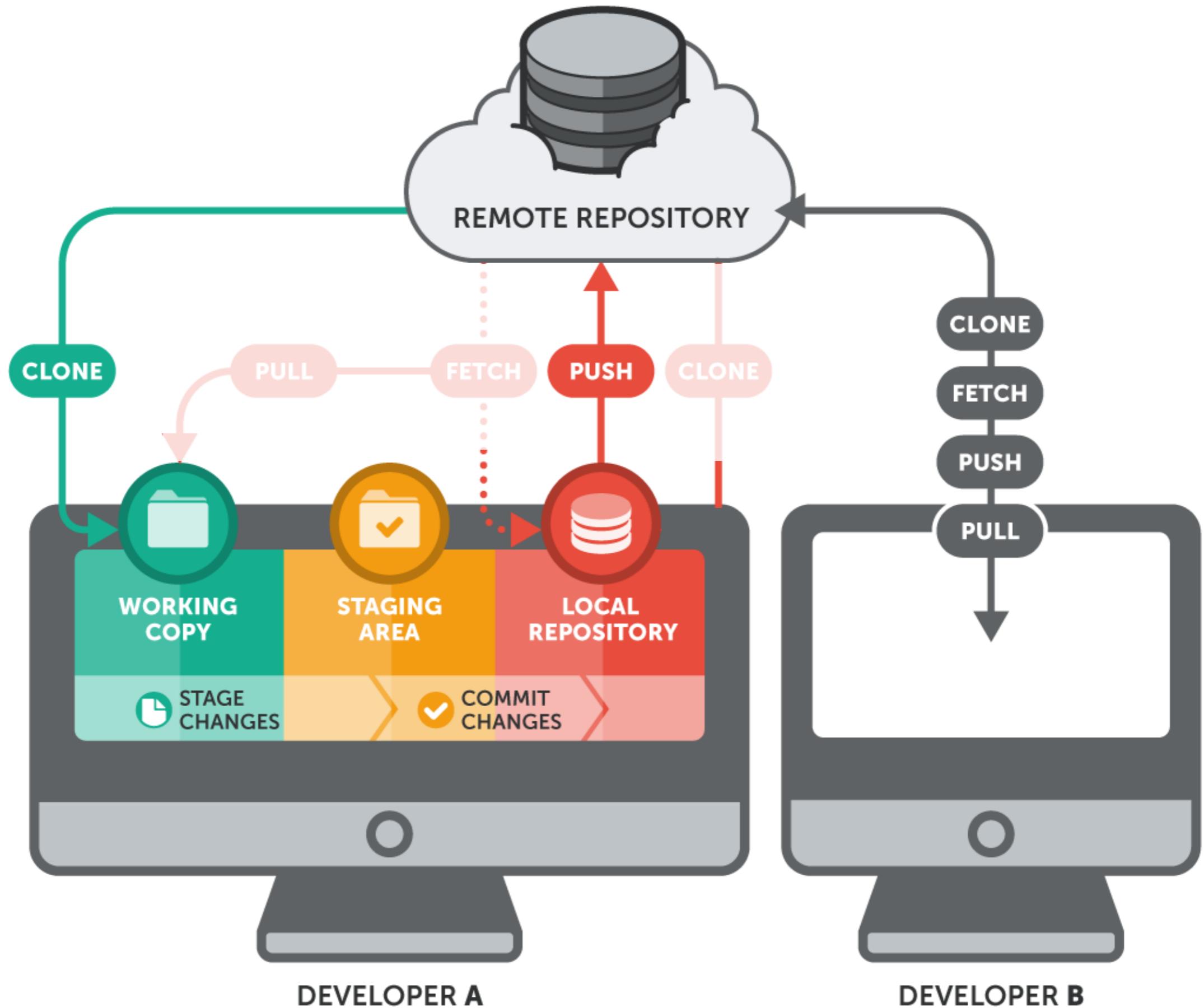
# Git the Main Idea:



# Git the Main Idea:

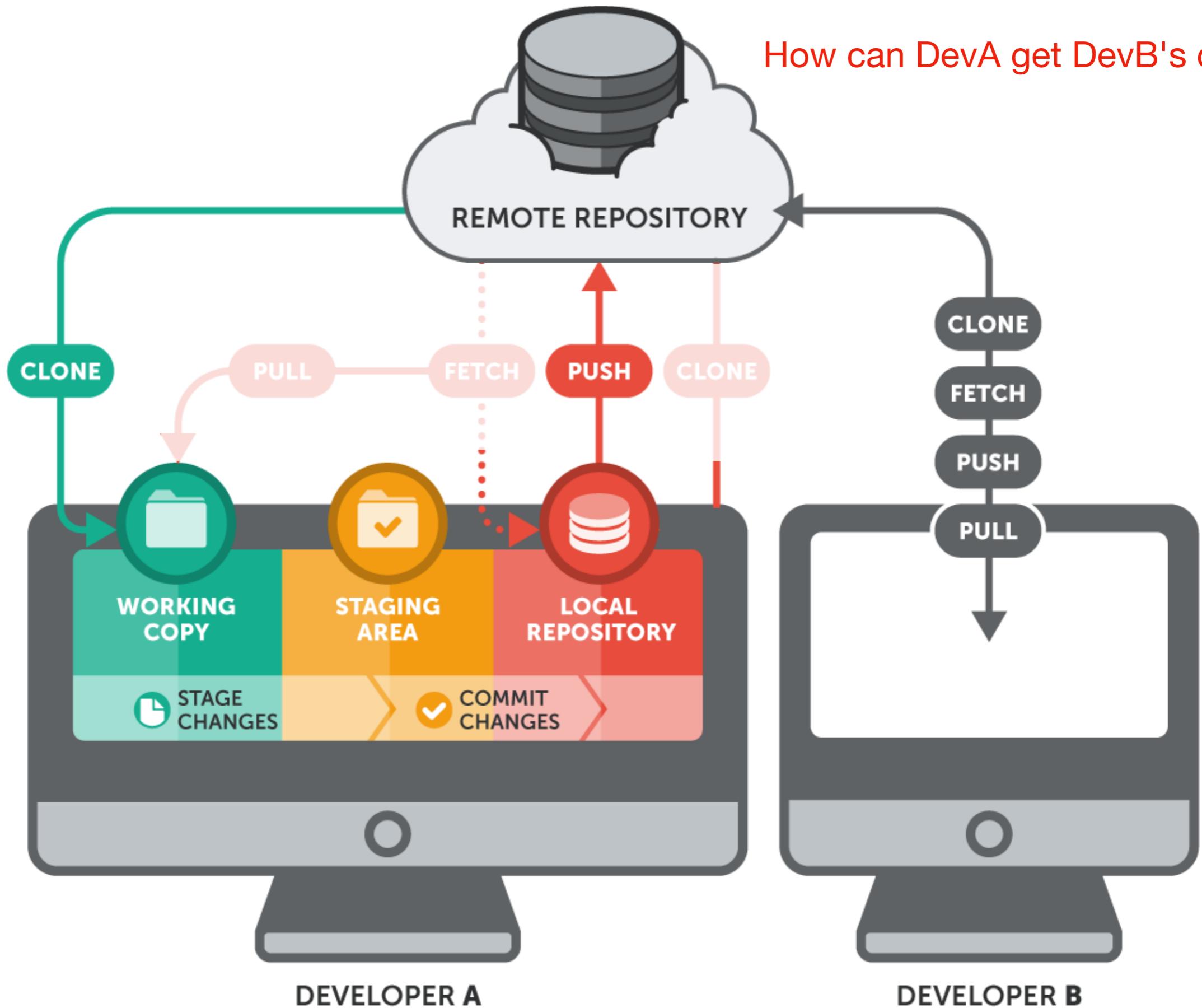


# Git the Main Idea:

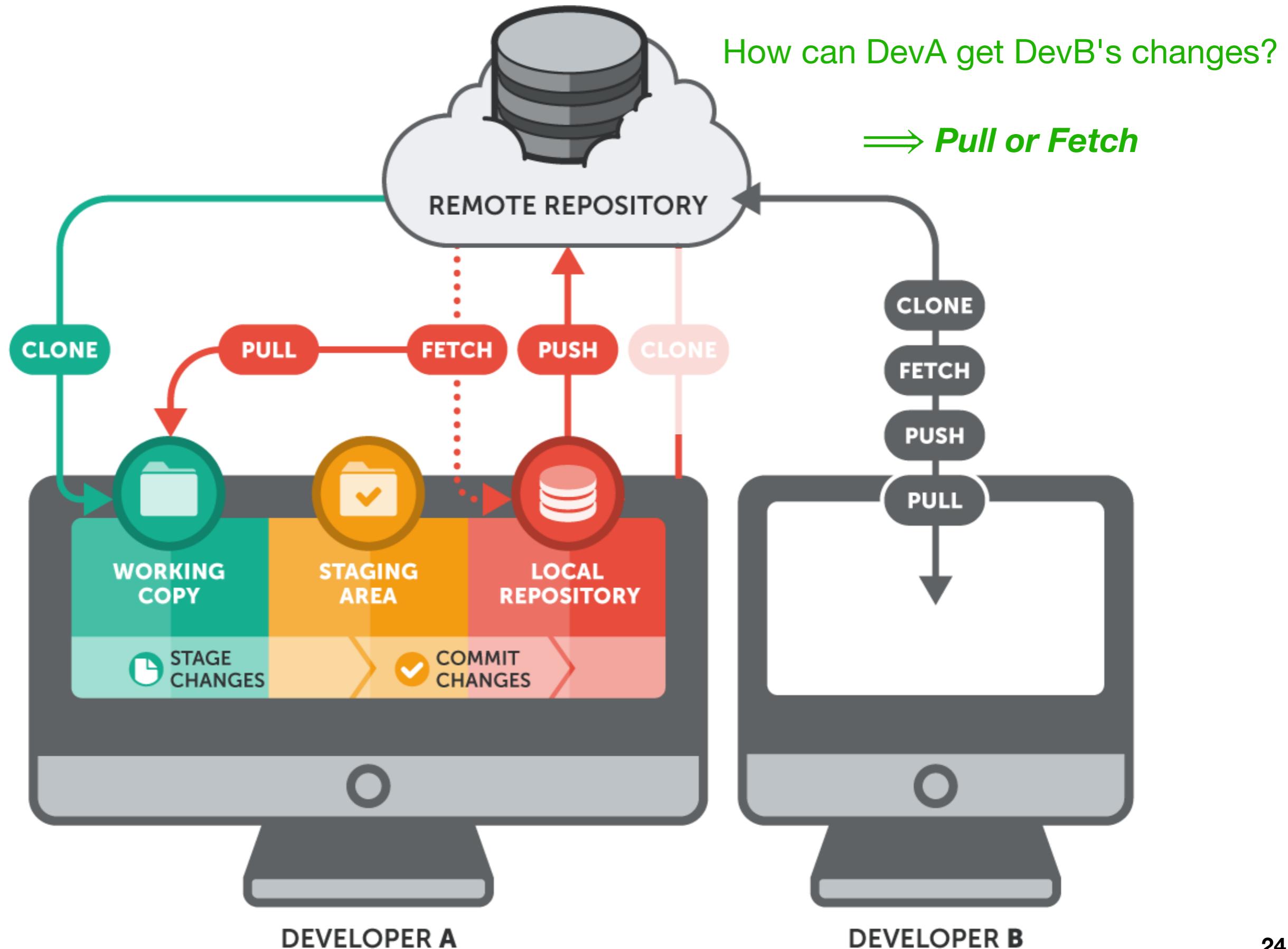


# Git the Main Idea:

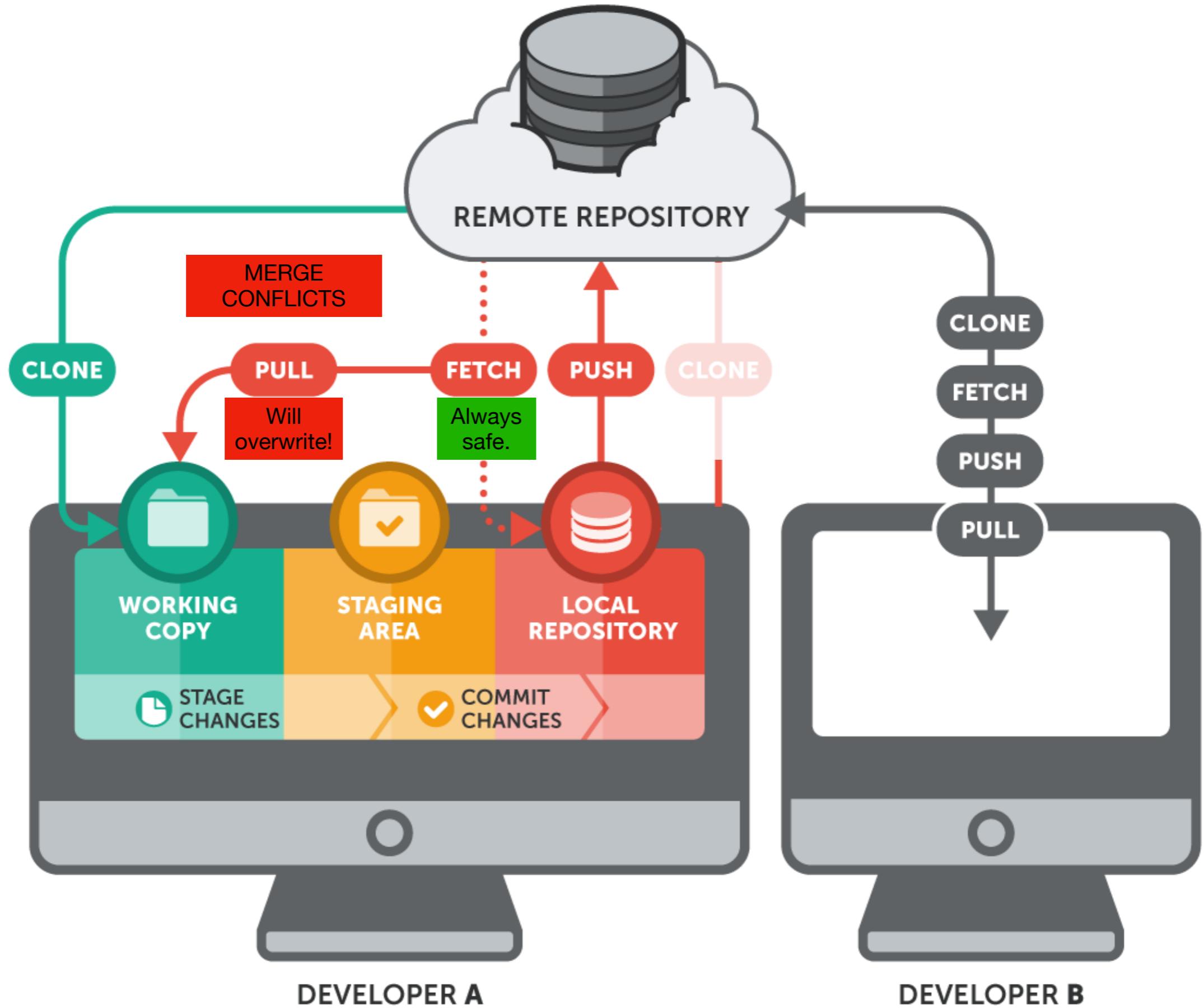
How can DevA get DevB's changes?



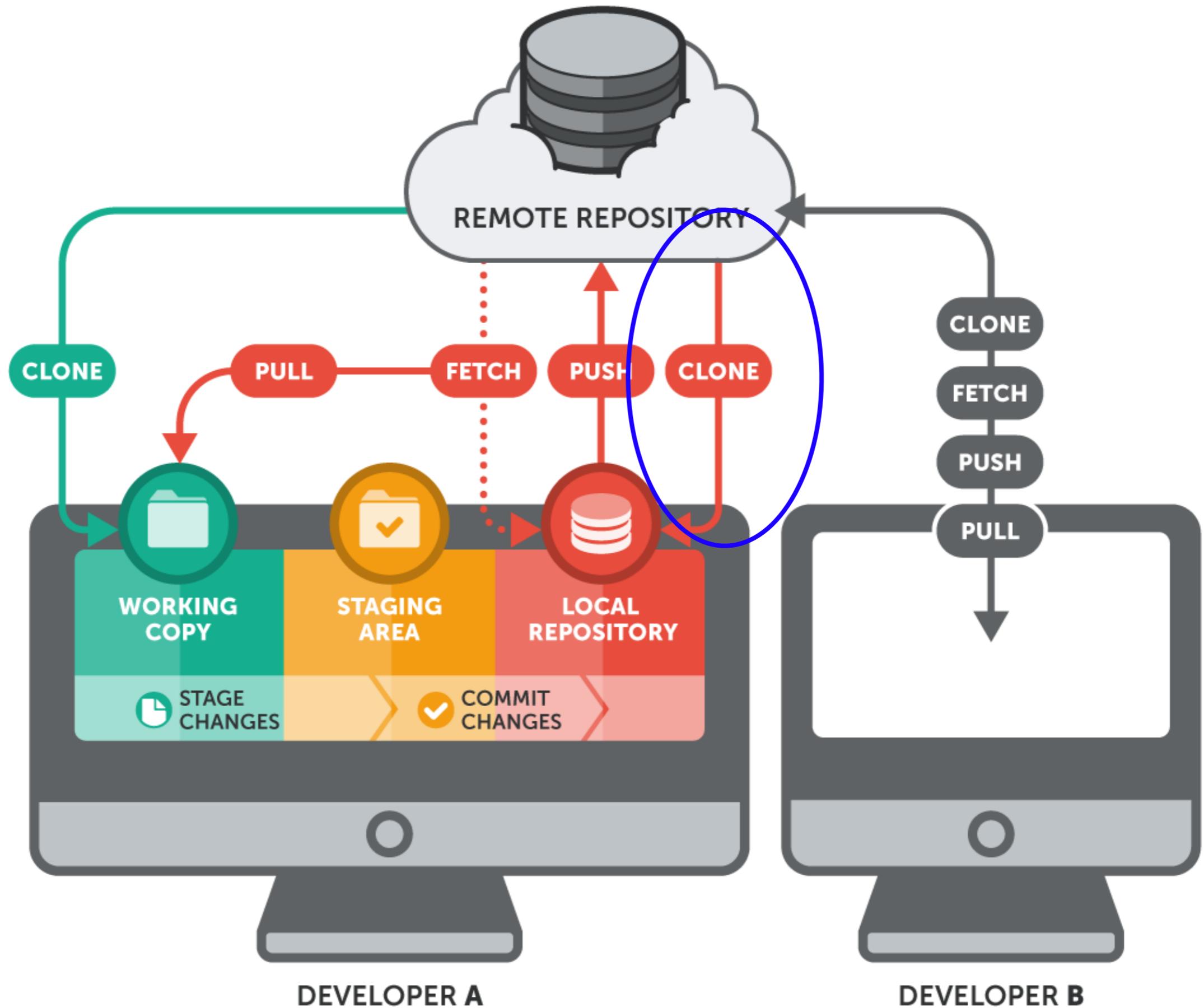
# Git the Main Idea:



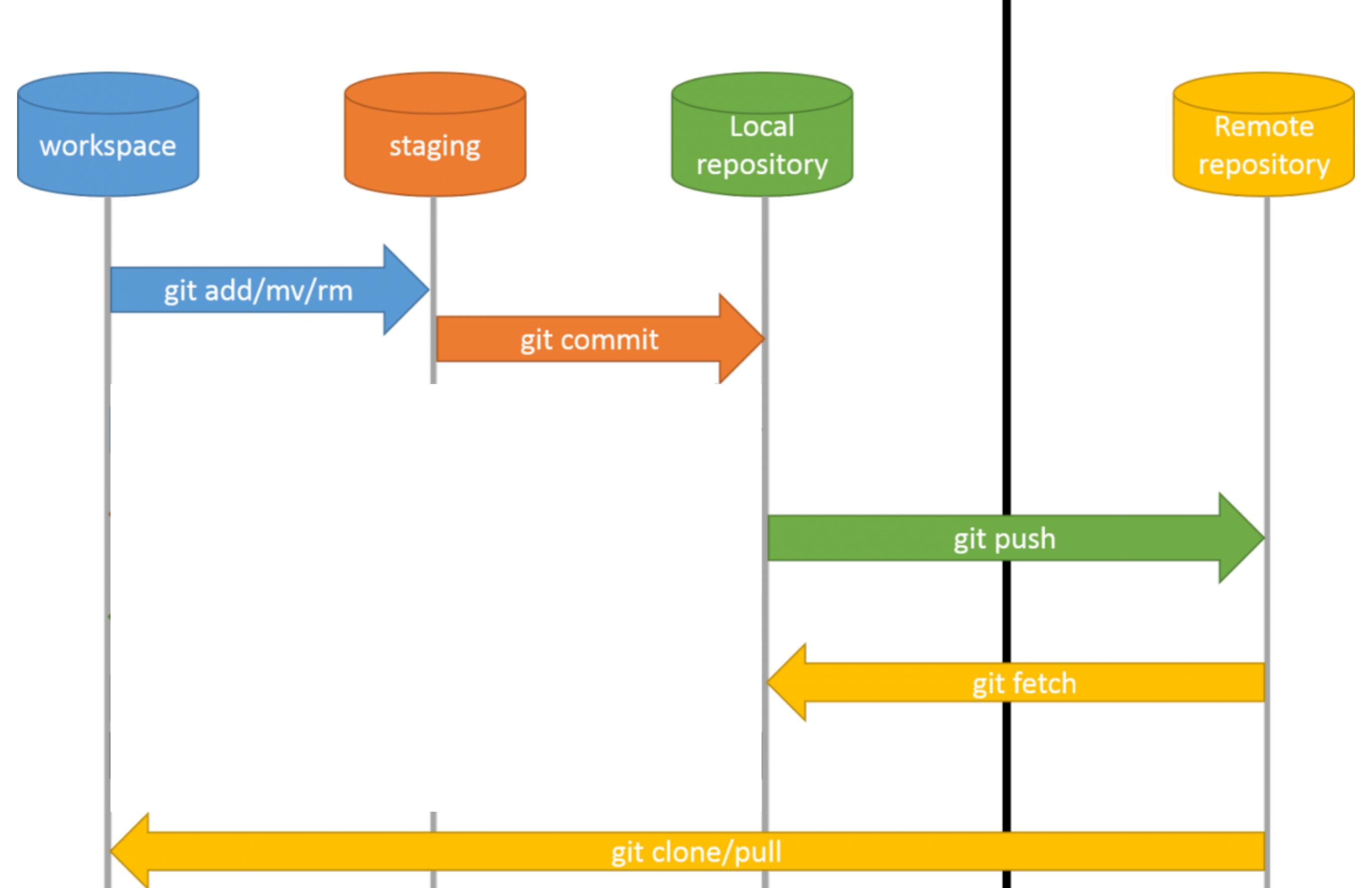
# Git the Main Idea:



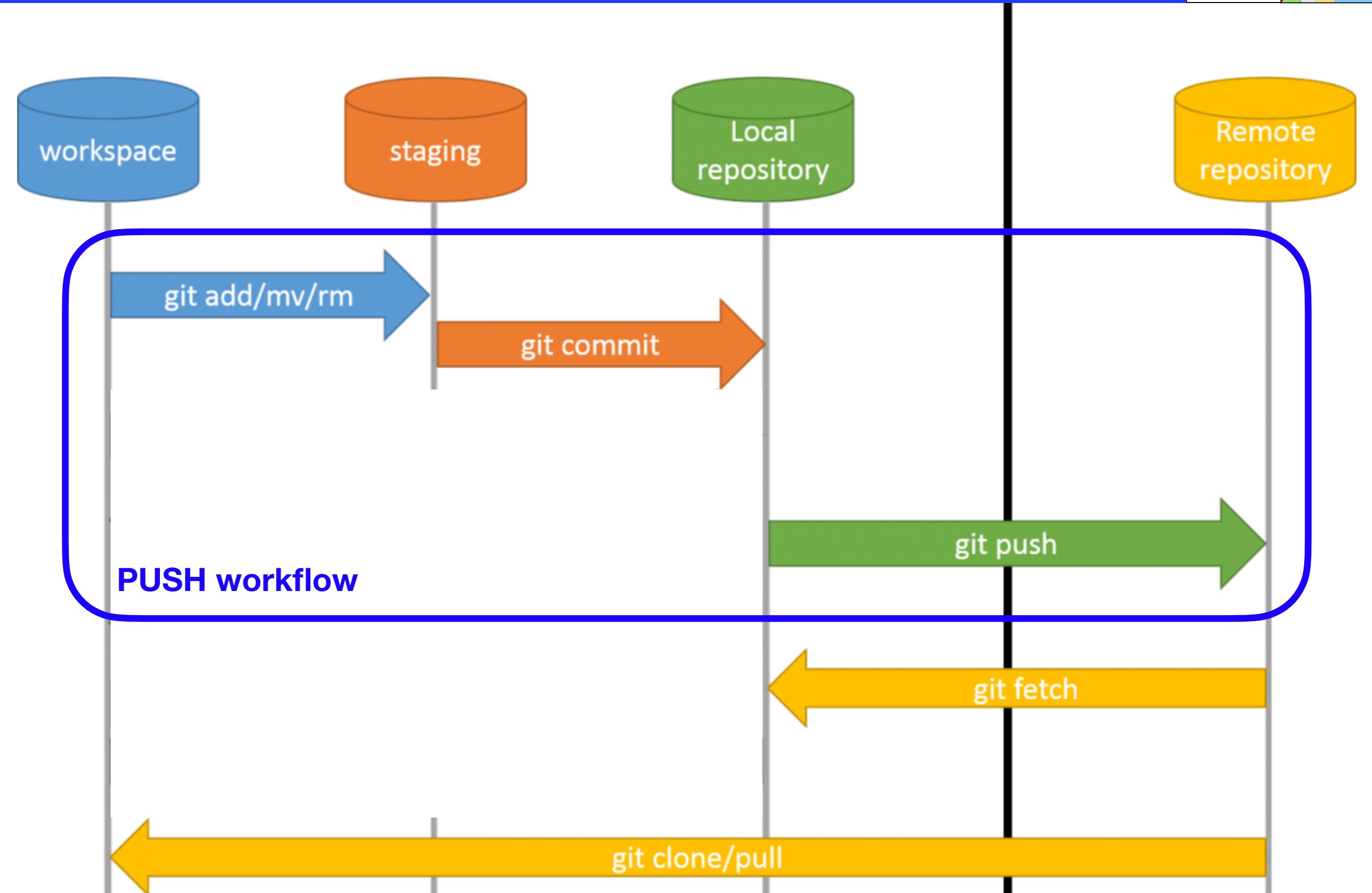
# Git the Main Idea:



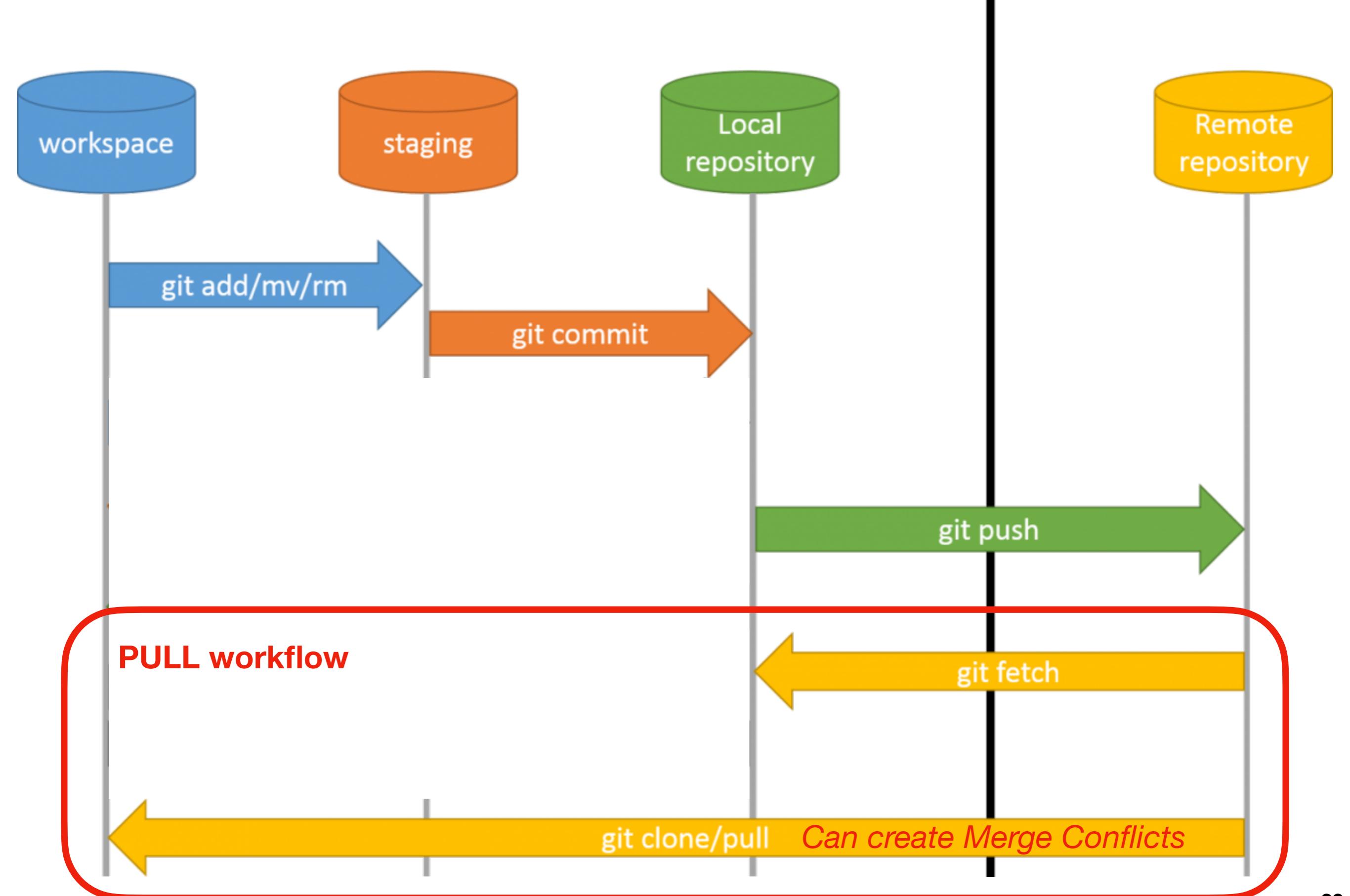
# The Push-Pull Workflow



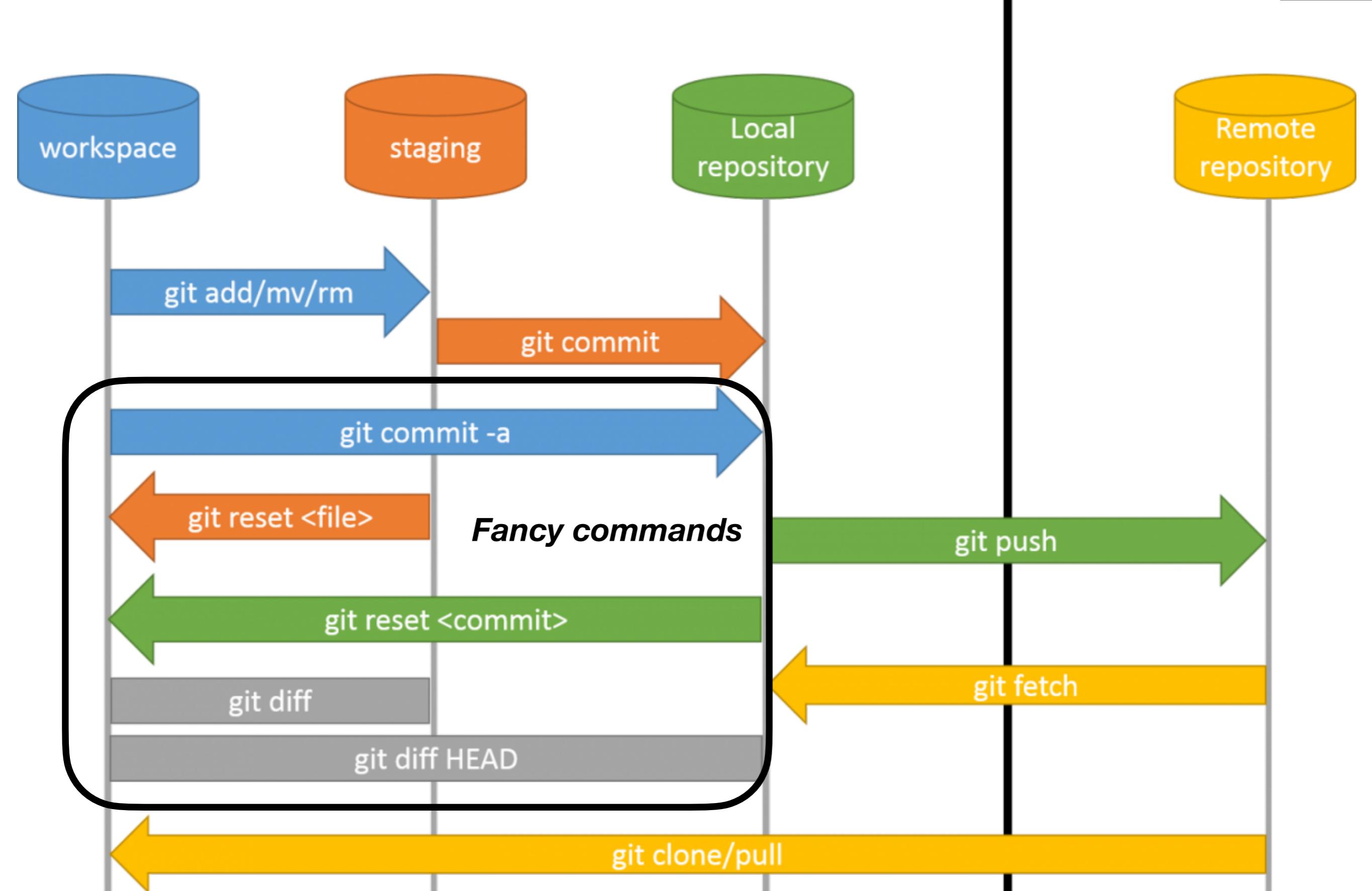
# The Push-Pull Workflow



# The Push-Pull Workflow

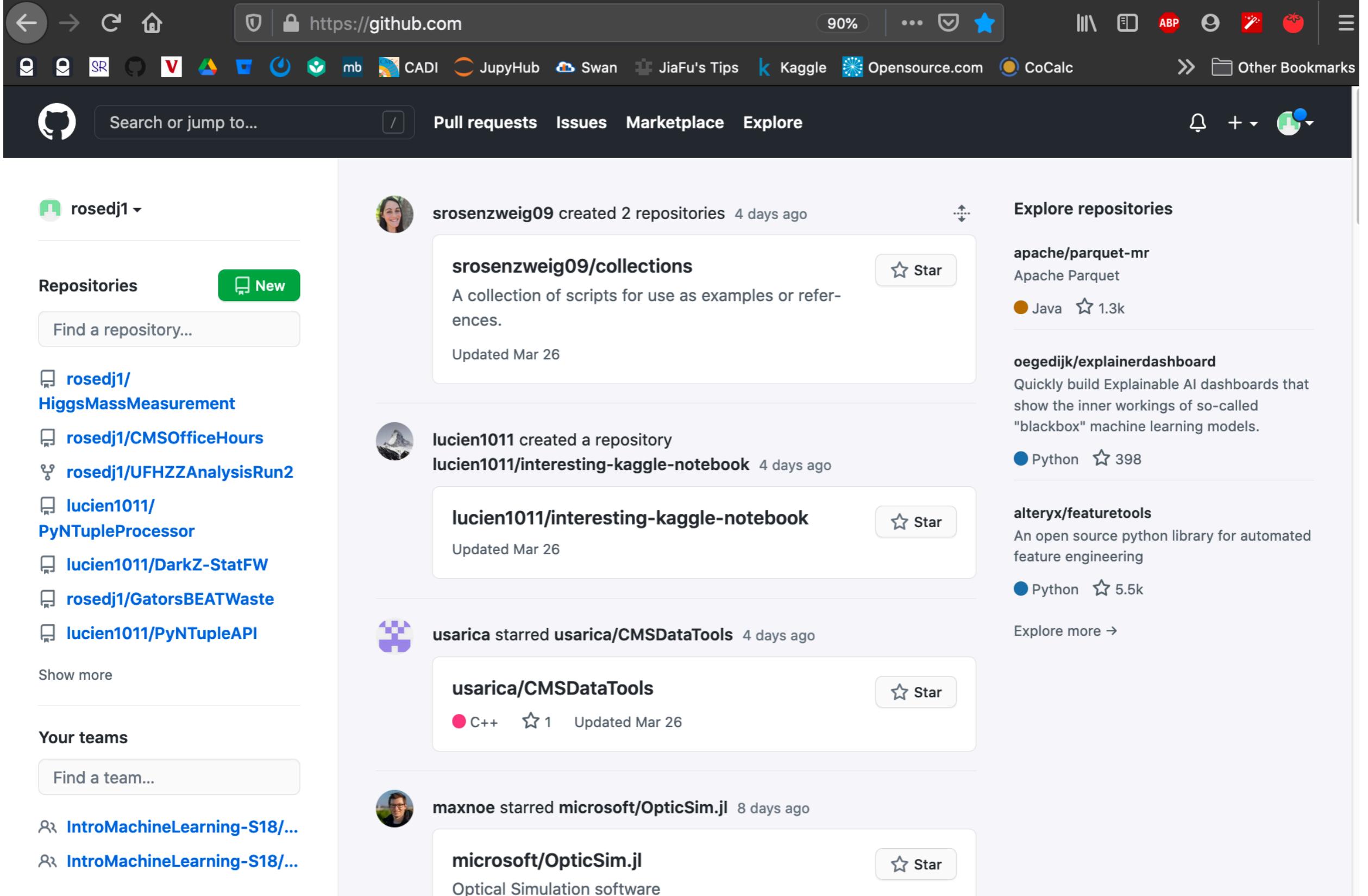


# The Push-Pull Workflow



# Let's practice! Make your own repo:

- Go to: <https://github.com/>



The screenshot shows a web browser window with the GitHub homepage loaded. The address bar at the top displays "https://github.com". The main content area shows several recent repository activity feeds. On the left sidebar, there are sections for "Repositories" (including "HiggsMassMeasurement", "CMSOfficeHours", etc.), "Your teams" (including "IntroMachineLearning-S18"), and a "Search or jump to..." input field. The central feed shows the following items:

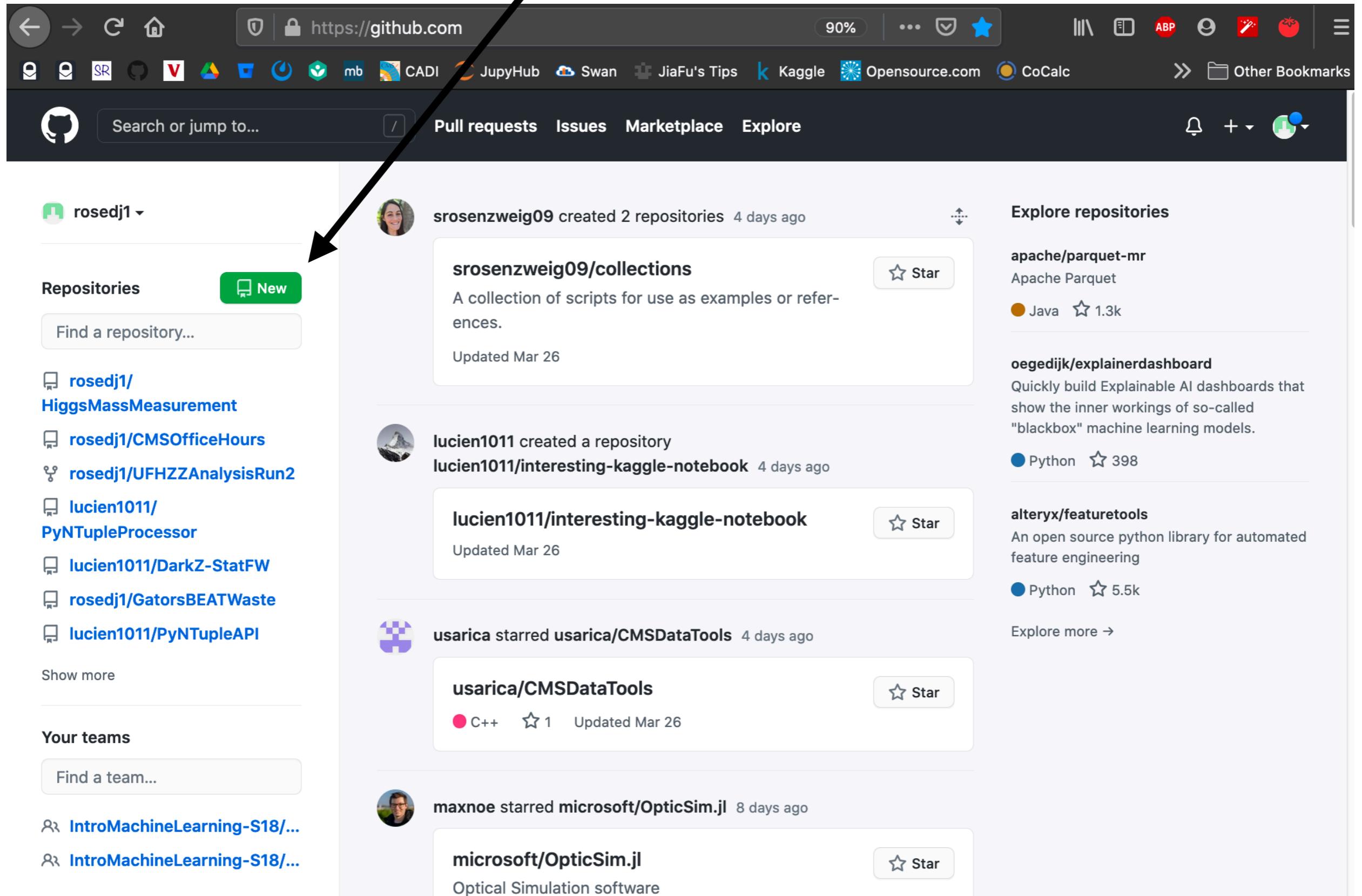
- srosenzweig09 created 2 repositories 4 days ago**
  - srosenzweig09/collections**: A collection of scripts for use as examples or references. Updated Mar 26. Star button.
  - lucien1011 created a repository lucien1011/interesting-kaggle-notebook 4 days ago**
    - lucien1011/interesting-kaggle-notebook**. Updated Mar 26. Star button.
- usrarica starred usrarica/CMSDataTools 4 days ago**
  - usrarica/CMSDataTools**. C++ 1. Updated Mar 26. Star button.
- maxnoe starred microsoft/OpticSim.jl 8 days ago**
  - microsoft/OpticSim.jl**: Optical Simulation software. Star button.

On the right side, there is a sidebar titled "Explore repositories" with links to other popular repositories like "apache/parquet-mr", "oe gedijk/explainerdashboard", and "alteryx/featuretools".

# Let's practice! Make your own repo:

- Go to: <https://github.com/>

Click the green 'New' Button



The screenshot shows the GitHub homepage. On the left, there is a sidebar with the user's profile picture and name (rosedj1), a 'Repositories' section listing several repositories like 'HiggsMassMeasurement', and a 'Your teams' section. In the center, there is a feed of recent repository activity. A large black arrow points from the text 'Click the green 'New' Button' towards the green 'New' button located in the top right corner of the 'Repositories' section. The feed includes entries such as 'srosenzweig09 created 2 repositories 4 days ago', 'lucien1011 created a repository lucien1011/interesting-kaggle-notebook 4 days ago', and 'maxnoe starred microsoft/OpticSim.jl 8 days ago'. On the right side, there is a sidebar titled 'Explore repositories' listing repositories like 'apache/parquet-mr', 'oegedijk/explainerdashboard', and 'alteryx/featuretools'.

# Let's practice! Make your own repo:

- Ideas for repos:

- ▶ Store your programming code
- ▶ Personal projects
- ▶ Company's work (Google, FB)
- ▶ Class/lecture notes, presentations, HW
- ▶ *Anything you want to save*
  - ◆ Nothing too large (<=50 MB/file)
  - ◆ No pictures, huge PDFs

## Create a new repository

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

**Owner \***      **Repository name \***

 rosedj1 / git-stuff ✓

Great repository names are short and memorable. Need inspiration? How about [animated-octo-waffle](#)?

**Description (optional)**

A tutorial repo for how to use Git.

 **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: Python ▾

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

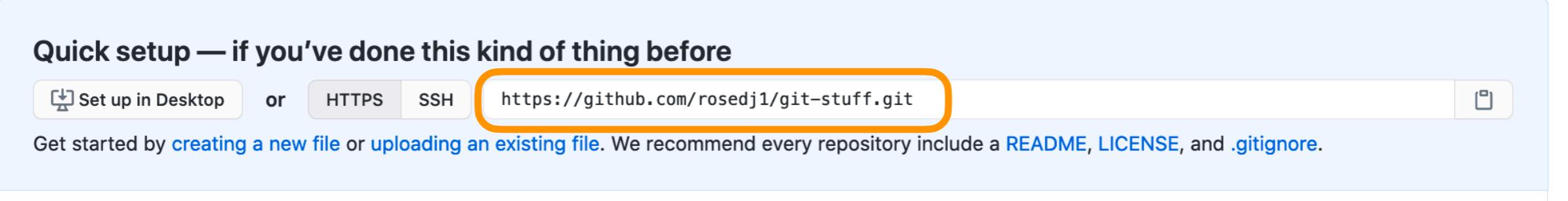
License: GNU General Public... ▾

This will set  `main` as the default branch. Change the default name in your [settings](#).

**Create repository**

# Let's practice! Make your own repo:

- Copy the URL:



A screenshot of the GitHub quick setup interface. At the top, it says "Quick setup — if you've done this kind of thing before". Below that are three buttons: "Set up in Desktop" (disabled), "or", "HTTPS", and "SSH". The "HTTPS" button is highlighted with a yellow border. To its right is a text input field containing the URL "https://github.com/rosedj1/git-stuff.git". On the far right of the input field is a small copy icon. Below the input field, there is a note: "Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#)".

# Let's practice! Make your own repo:

- Copy the URL:

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

[!\[\]\(44c16be7d76e03a4be936eed0a777d82\_img.jpg\) Set up in Desktop](#) or [!\[\]\(de7c45071f9bfe76c47034f87707394a\_img.jpg\) HTTPS](https://github.com/rosedj1/git-stuff.git) [!\[\]\(4eeefa738baae81fe658a7eeb675878f\_img.jpg\) SSH](#) [!\[\]\(b8e9770ac101e4955db03ac3ca0babc1\_img.jpg\) https://github.com/rosedj1/git-stuff.git](https://github.com/rosedj1/git-stuff.git)

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

- Clone it to your local machine: `git clone https://...`

```
(base) Jakes-MacBook-Pro:$ mkdir ~/Practice
(base) Jakes-MacBook-Pro:$ cd !$
cd ~/Practice
(base) Jakes-MacBook-Pro:/Users/Jake/Practice/$ git clone https://github.com/rosedj1/git-stuff.git
Cloning into 'git-stuff'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (5/5), done.
(base) Jakes-MacBook-Pro:/Users/Jake/Practice/$ ls
total 0
drwxr-xr-x  6 Jake  staff  192B Mar 30 15:14 git-stuff ← Your local version of the remote repo
(base) Jakes-MacBook-Pro:/Users/Jake/Practice/$ cd git-stuff/
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ pwd
/Users/Jake/Practice/git-stuff
```

- Anything added inside your local repo can be tracked by `git`.

# Let's practice! Add some code:

- Make some code in your local repo:

```
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ echo "import numpy" > newcode.py
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ cat newcode.py
import numpy
```

# Let's practice! Add some code:

- Make some code in your local repo:

```
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ echo "import numpy" > newcode.py
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ cat newcode.py
import numpy
```

- What does Git currently know about?

```
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    newcode.py

nothing added to commit but untracked files present (use "git add" to track)
```

# Let's practice! Add some code:

- Make some code in your local repo:

```
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ echo "import numpy" > newcode.py
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ cat newcode.py
import numpy
```

- What does Git currently know about?

which branch  
you're on

Git sees this  
new code  
but won't  
track it yet

```
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ git status
On branch main
Your branch is up to date with 'origin/main'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    newcode.py
nothing added to commit but untracked files present (use "git add" to track)
```

The nickname of your remote repo

- Use git status often! *Informative.*

# Let's practice! Add some code:

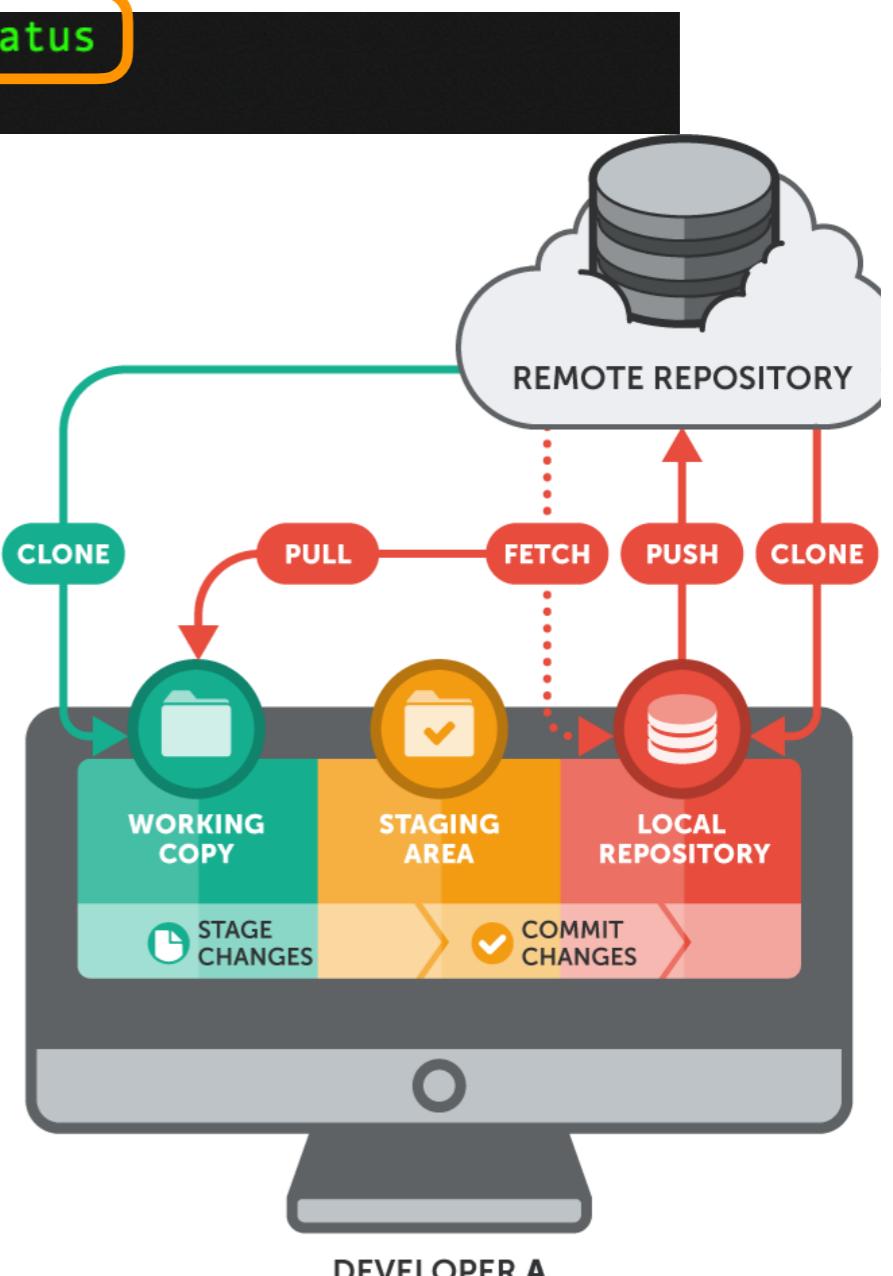
- Make some code in your local repo:

```
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ echo "import numpy" > newcode.py
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ cat newcode.py
import numpy
```

- What does Git currently know about?

which branch you're on → (base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/\$ git status  
 On branch main  
 Your branch is up to date with origin/main'.  
 The nickname of your remote repo  
 Untracked files:  
 (use "git add <file>..." to include in what will be comm  
 newcode.py  
 nothing added to commit but untracked files present (use "

Git sees this new code but won't track it yet →



- Use git status often! *Informative*.
- Q: In which part of the workflow do **untracked files** live?

# Let's practice! Add some code:

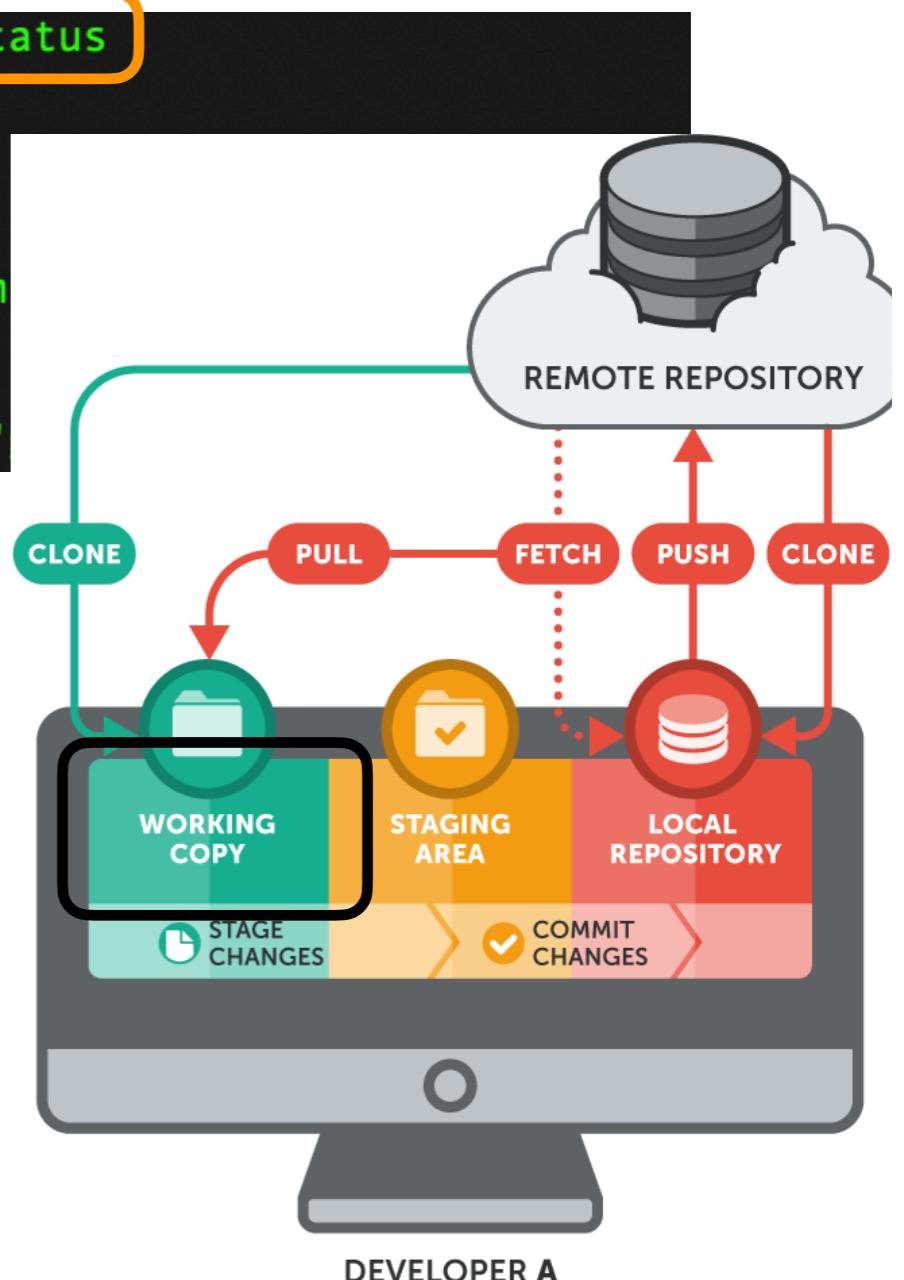
- Make some code in your local repo:

```
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ echo "import numpy" > newcode.py
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ cat newcode.py
import numpy
```

- What does Git currently know about?

which branch you're on → (base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/\$ git status  
 On branch main  
 Your branch is up to date with origin/main'.  
 The nickname of your remote repo  
 Untracked files:  
 (use "git add <file>..." to include in what will be comm  
 newcode.py  
 nothing added to commit but untracked files present (use "

Git sees this new code but won't track it yet →



- Use git status often! *Informative*.
- Q: In which part of the workflow do **untracked files** live?

Untracked files live in your Working Area

# Let's practice! Add some code:

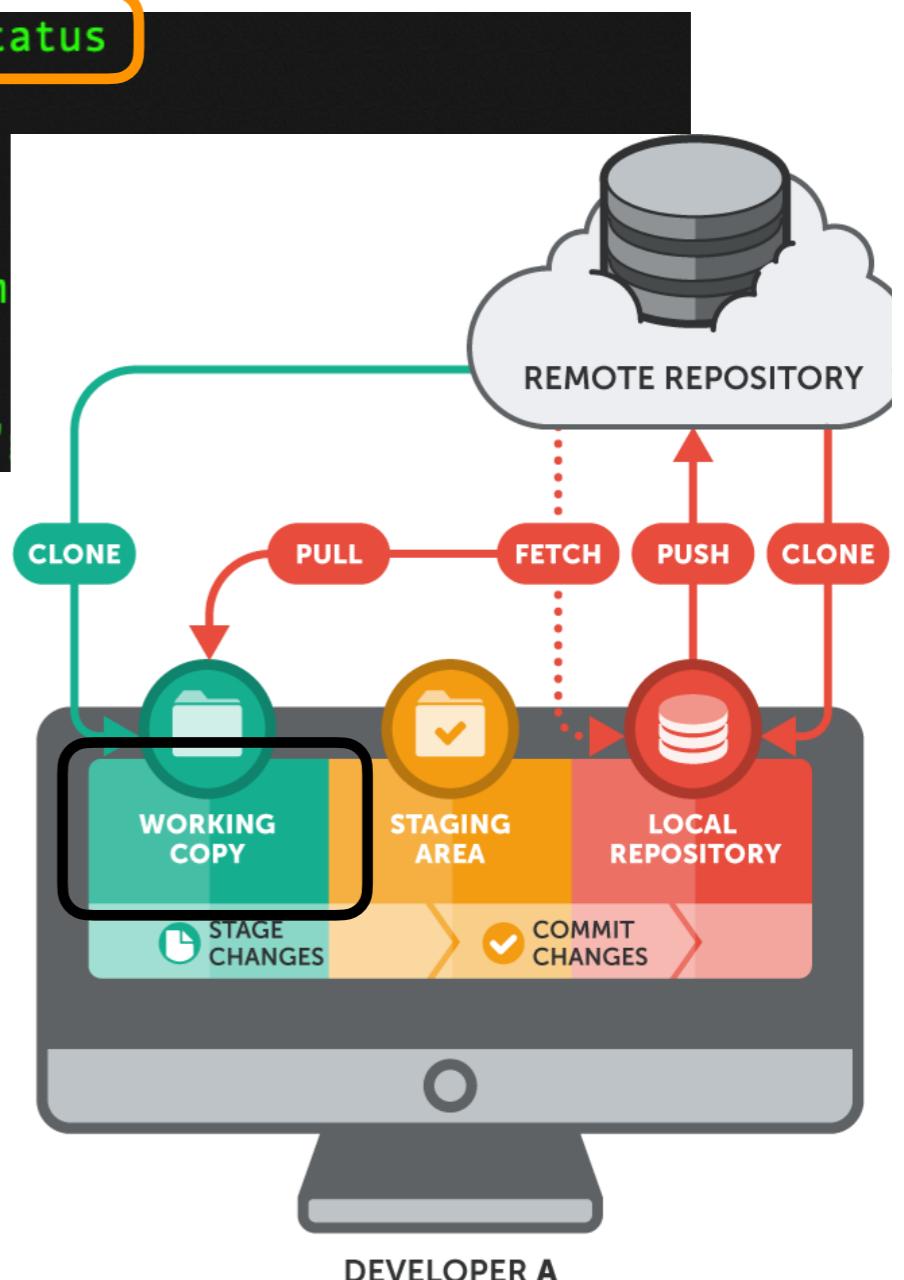
- Make some code in your local repo:

```
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ echo "import numpy" > newcode.py
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ cat newcode.py
import numpy
```

- What does Git currently know about?

which branch you're on → (base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/\$ git status  
 On branch main  
 Your branch is up to date with origin/main'.  
 The nickname of your remote repo  
 Untracked files:  
 (use "git add <file>..." to include in what will be committed)  
 newcode.py  
 nothing added to commit but untracked files present (use "git add" to track changes)

Git sees this new code but won't track it yet →



- Use git status often! *Informative*.
- Q: In which part of the workflow do **untracked files** live?
- Q: How do we add untracked files to the **Staging Area**?

Untracked files live in your Working Area

# Let's practice! Add MORE code:

- Q: How do we add untracked files to the **Staging Area**?

► git add <file>

```
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   newcode.py
```

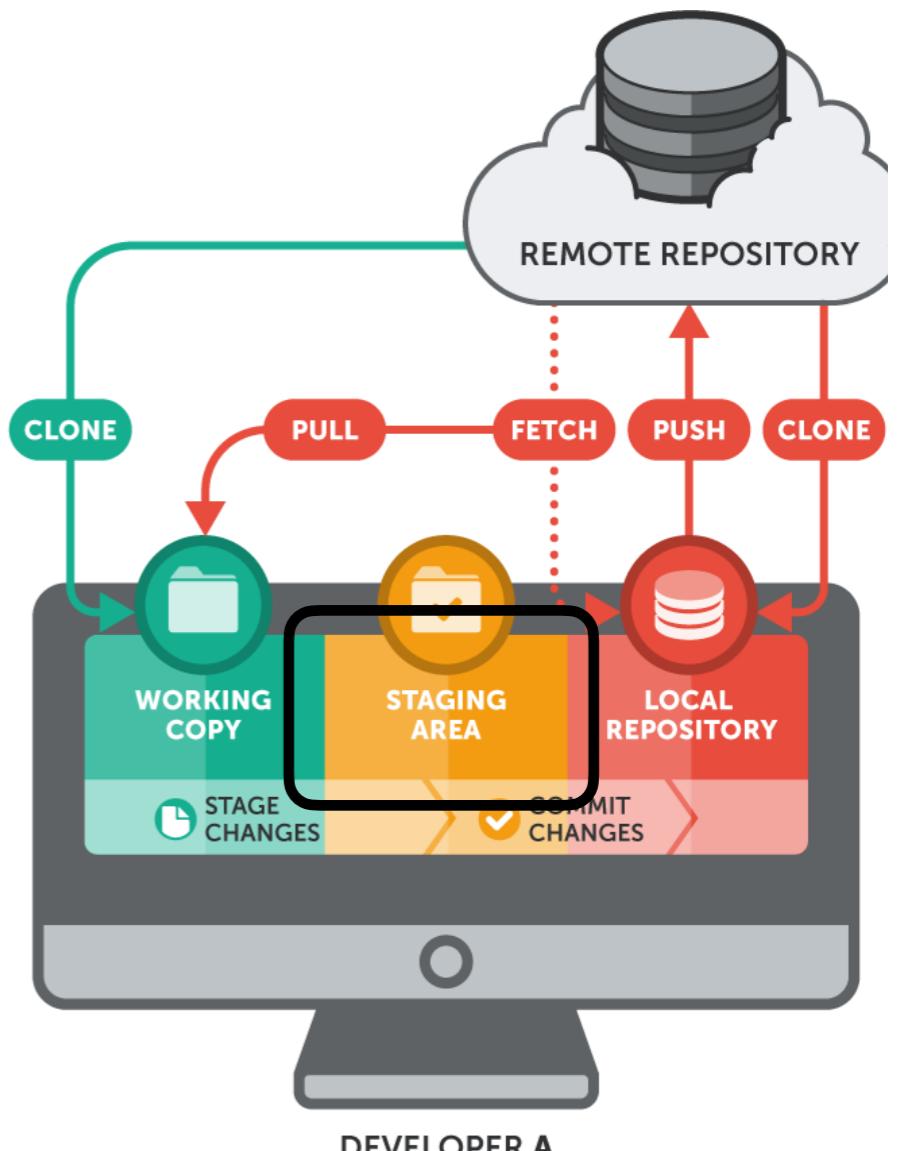
- Now this file is **staged**.

► Git is ready to commit. *Are you?*

- *Don't commit yet!*

Create another file called "second.txt".  
Then `git status`.

Added files  
live in your  
**Staging Area**



```
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ echo "some words" > second.txt
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   newcode.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    second.txt
```

Q: *What is this Git message telling us?*

newcode.py has **already been staged**.  
It is ready to be **committed** (saved).

But second.txt is **untracked** (new).  
This file is still in the Working Area.

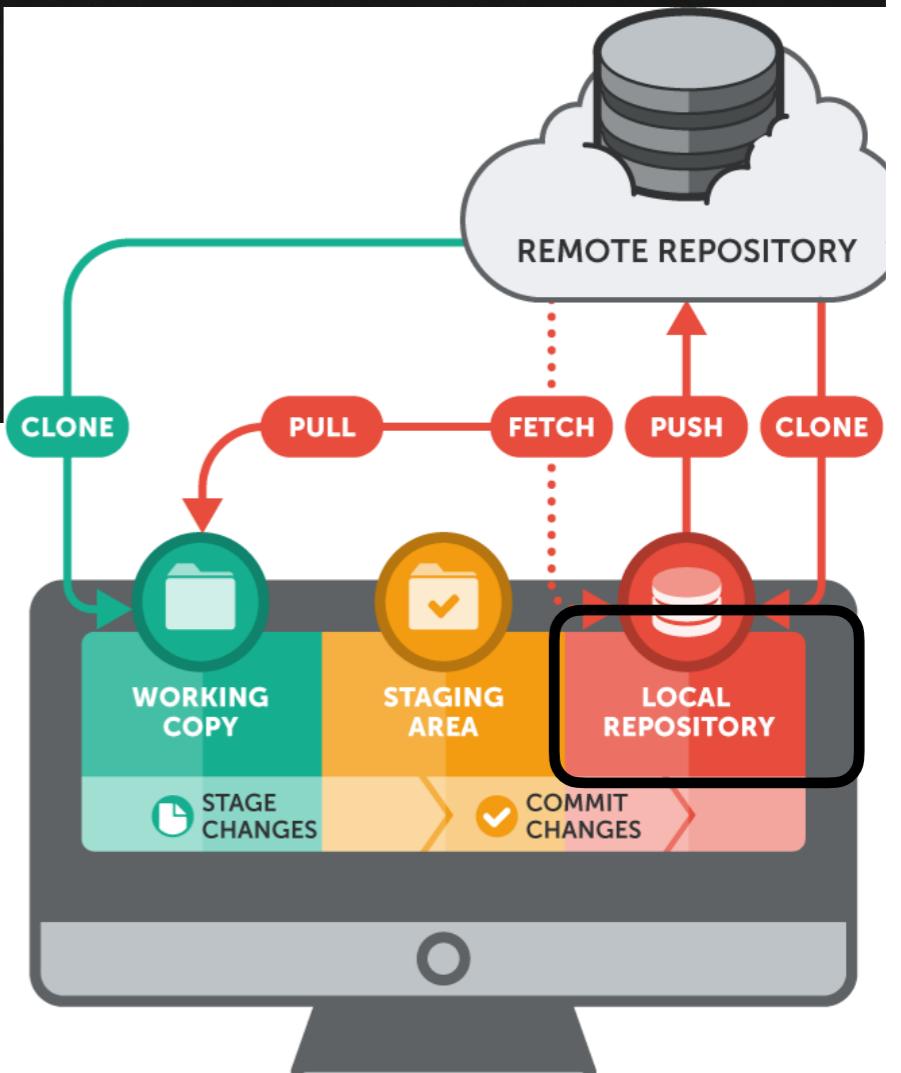
# Let's practice! Commit your code:

```
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ git commit -m "Added new code for testing." -m "It simply imports numpy."
[main 2638c0b] Added new code for testing.
 1 file changed, 1 insertion(+)
 create mode 100644 newcode.py
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    second.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Primary message      Secondary message

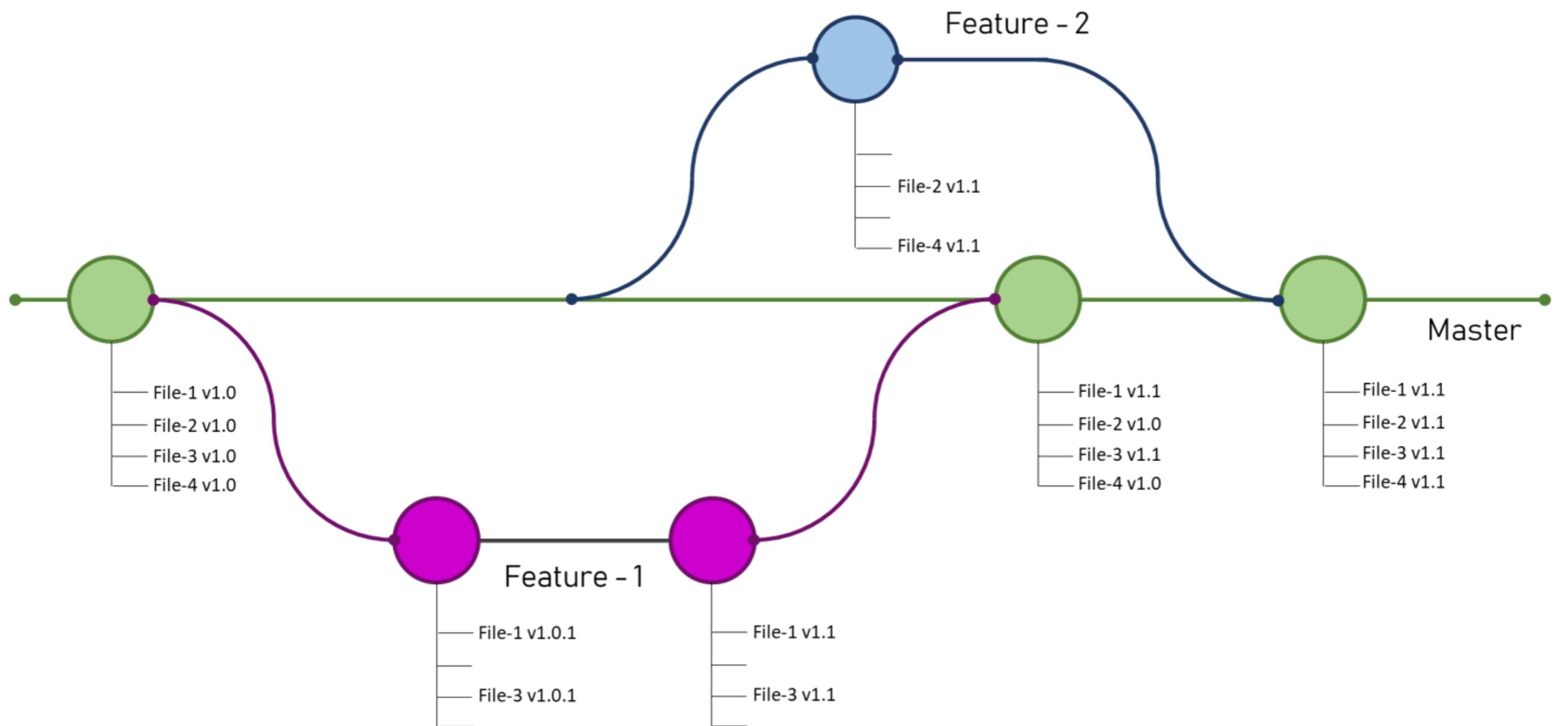


- Now your code is saved in your **Local Repo!**
- Q: How do make this local code talk to your remote repo?
  - ▶ Push it up!

```
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ git branch
* main
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ git remote
origin
(base) Jakes-MacBook-Pro:/Jake/Practice/git-stuff/$ git push -u origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 310 bytes | 310.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/rosedj1/git-stuff.git
  414df0b..2638c0b  main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

# Branches

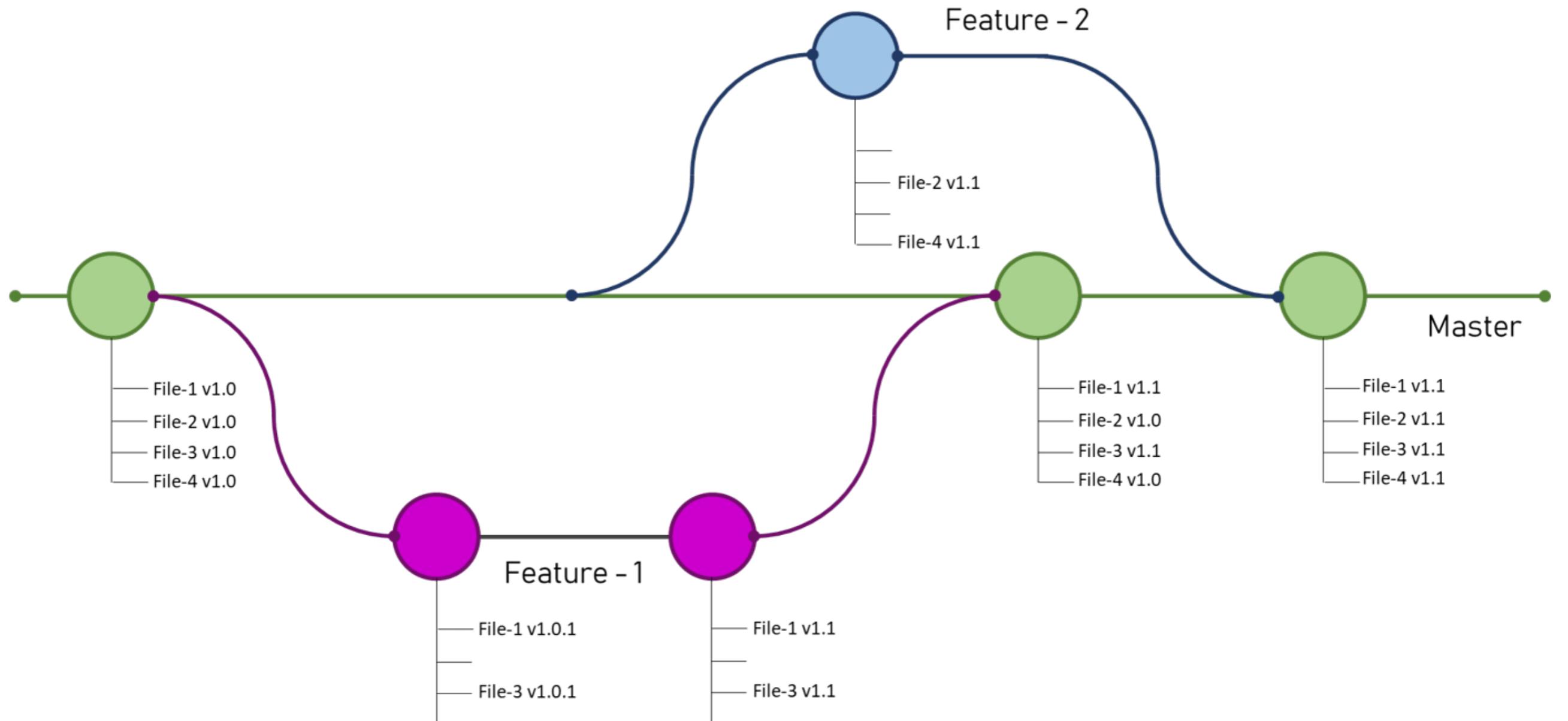
- So far you've been working on a branch called "**main**" (master)



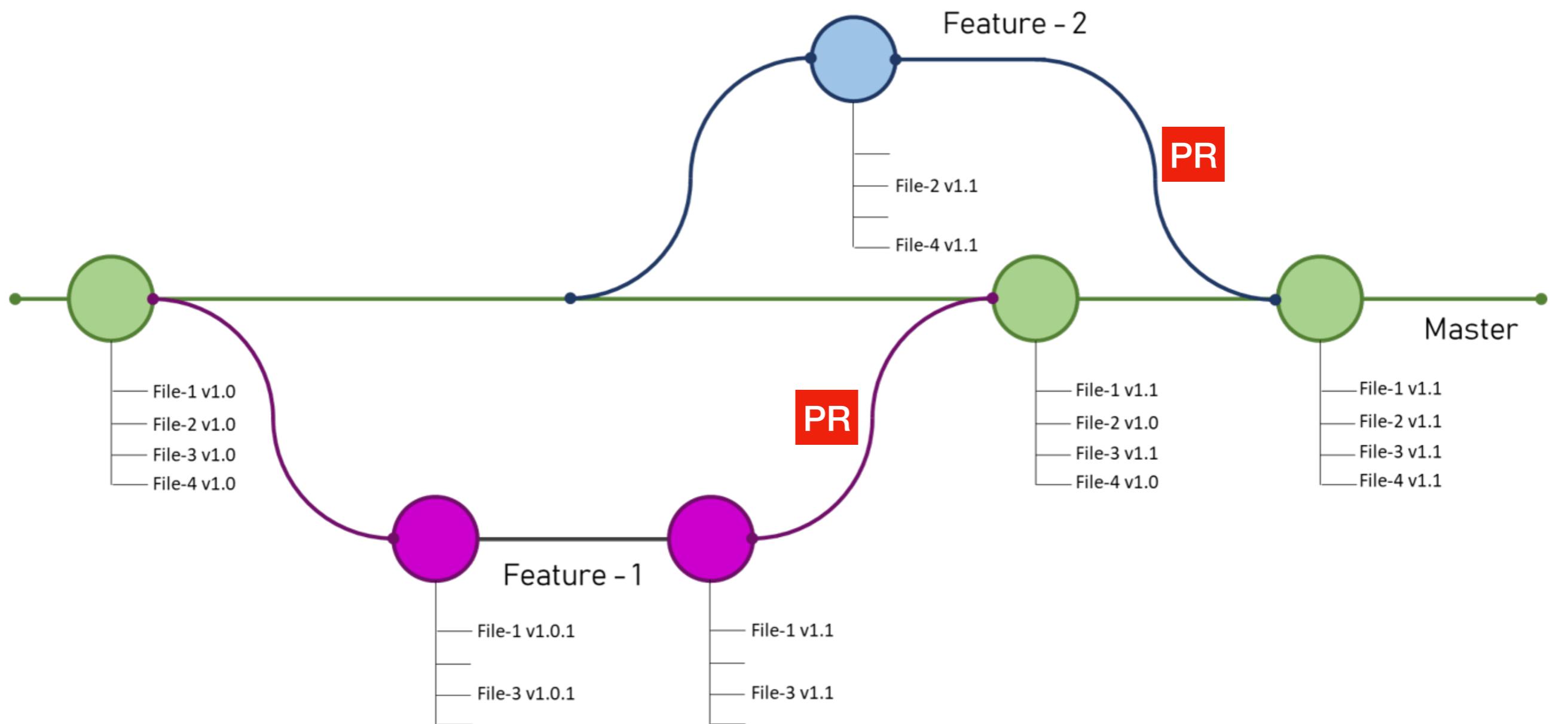
# Branches

- So far you've been working on a branch called "**main**" (master)
- It's a better idea to work on your own branch: "**Feature-1**"

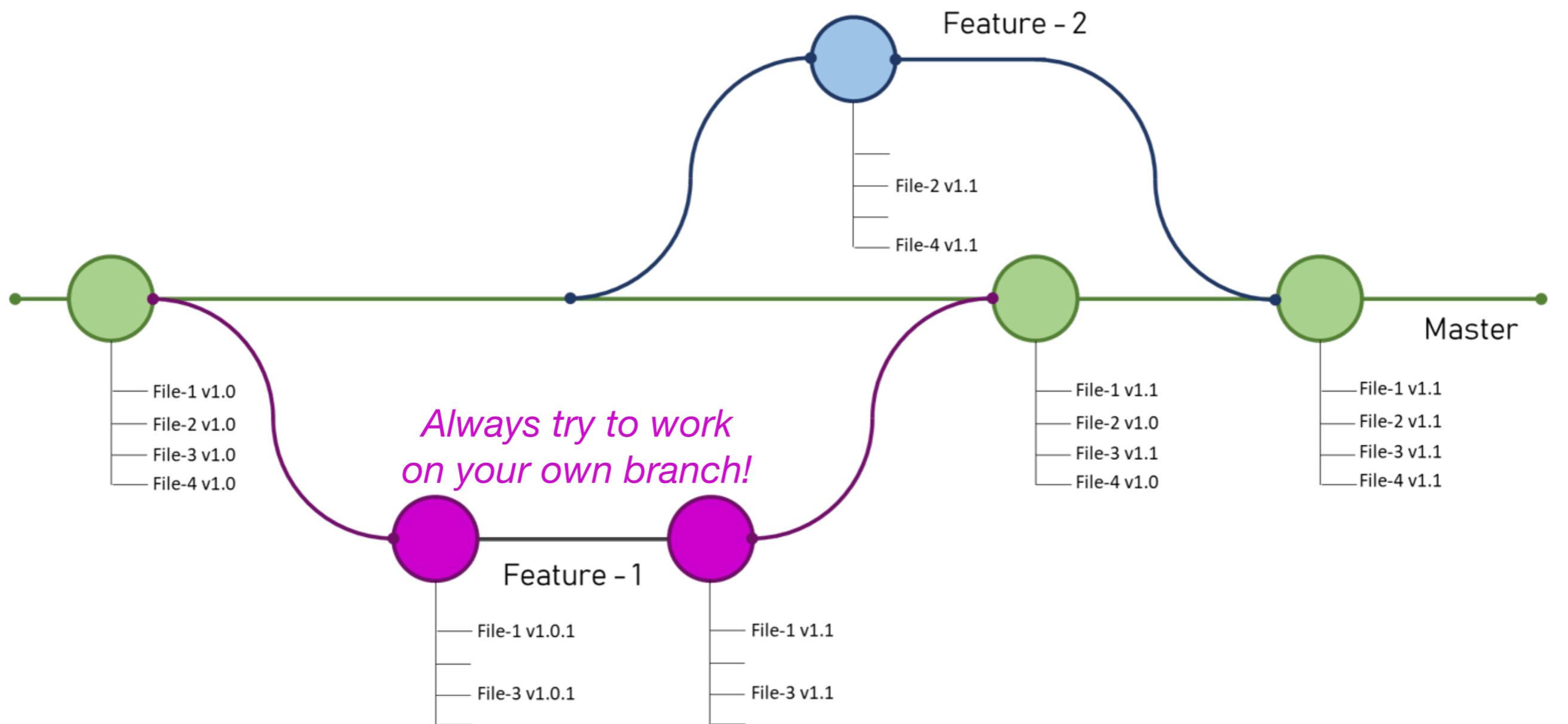
`git checkout -b Feature-1`



# Branches: Pull Requests merge branches



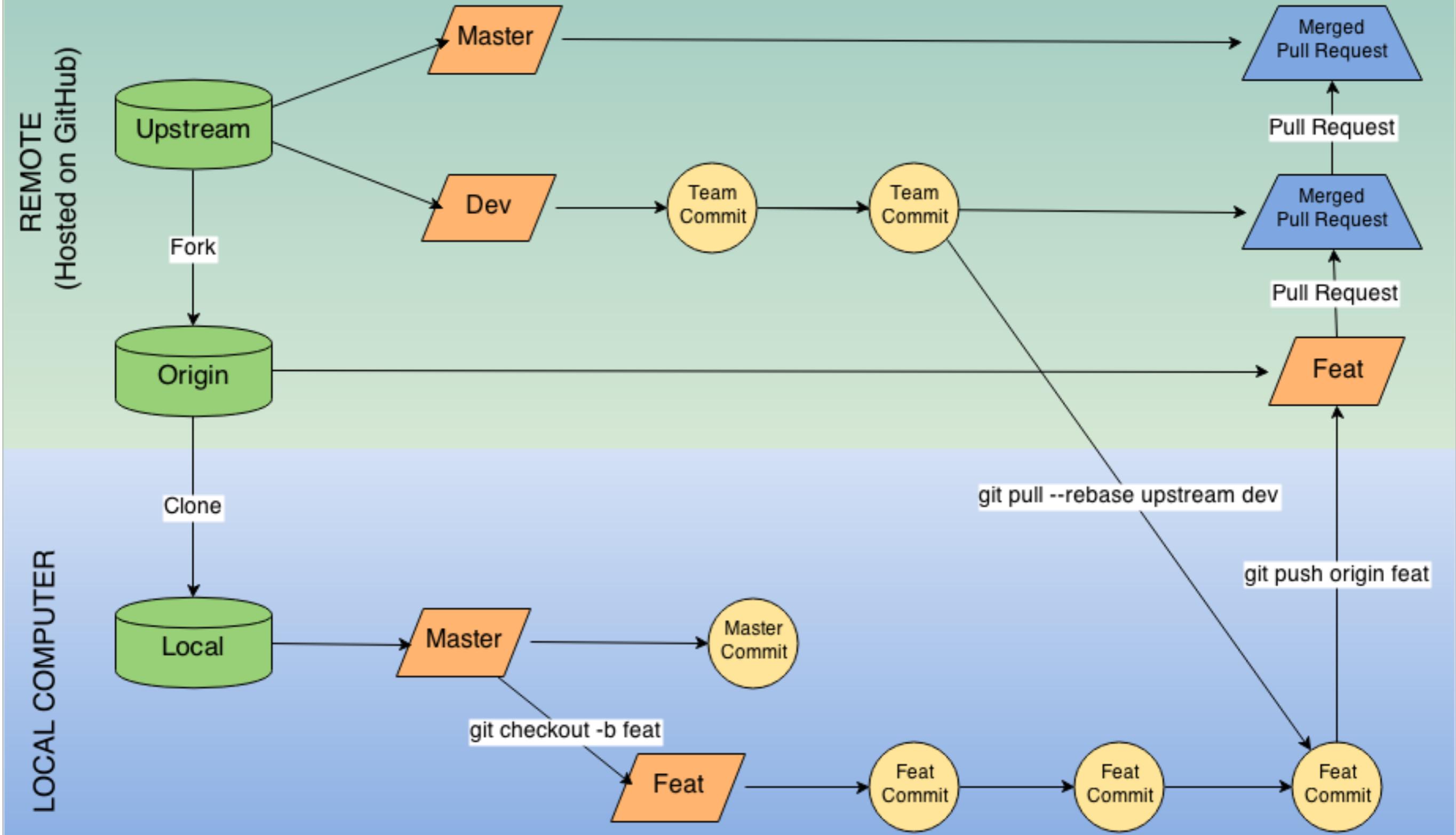
# Branches: Use your own branch



# Redo Task 1, but do it the safe way

- Make your own branch
- Make new code
- Then Add, Commit, and Push to remote repo:
  - Add: `git add <newestfile.txt>`
  - Commit: `git commit -m "Some descriptive message."`
  - Push: `git push origin feature-1`
- Make a PR.
- Merge PR.

# Git Workflow



# Other commands

- **git reset --hard** //Used to reset your repo and delete all commits tracked by GitHub so if you pull it will completely overwrite everything and will not try to merge.
- **git stash** //git stash will store your local changes without pushing them to the repo
- **git stash pop** //apply the changes you stashed again
- **git log --graph --oneline**
- **git diff file1..file2**
- **git diff file1...file2**

Now you know  
these commands

Essential git commands every developer  
should know

Git Clone  
Git branch  
Git checkout  
Git status  
Git add  
Git commit  
Git push  
Git pull  
Git revert  
Git merge  
Git remote  
Git fetch



# Git Tips:

- Push often!
  - ▶ Like every day
- Don't work on the main branch
  - ▶ Use your own branch
  - ▶ Push to remote
  - ▶ Make PR to merge your branch into main
- Use detailed messages
  - ▶ Not only in commit messages, but in PRs, etc.

# Let's practice! Task 2: Fork this repo

- In terminal, go to your home area:   
▶ `cd ~`
- Point to CMS OH repo:
  - ▶ Go to: <https://github.com/rosedj1/>
  - ▶ Click **CMS\_Office\_Hours**
  - ▶ Click green button
  - ▶ Copy HTTPS link
- Clone remote repo to your **local area**
  - ▶ This is YOUR local repo!
  - ▶ Will not affect the remote repo (unless you tell it to)
- Make (checkout) a new branch
  - ▶ Do not work on the master branch!
- Begin creating new code!
  - ▶ `git clone https://...`
  - ▶ `git checkout -b <mynewbranch>`
- Now Sean's changes are on HIS remote repo on his own branch (not master)
  - ▶ Begin creating new code!
- Do PR into origin/master
- Everyone else can pull down Sean's changes.

# Backup Slides

# Commits

- A commit is a **snapshot of your files**

# git diff

- Just show browser diffs



# Useful branch commands

- `git branch -a`
  - ▶ Shows all branches
- `git branch -r`
  - ▶ Show all remote branches
- `git branch -v`
  - ▶ verbose
- `git branch -h`
  - ▶ See help options.
- `git checkout -b mynewbranch origin/onlinebranch`
  - ▶ Checkout onlinebranch that exists on the remote repo and call it mynewbranch

# Terminology

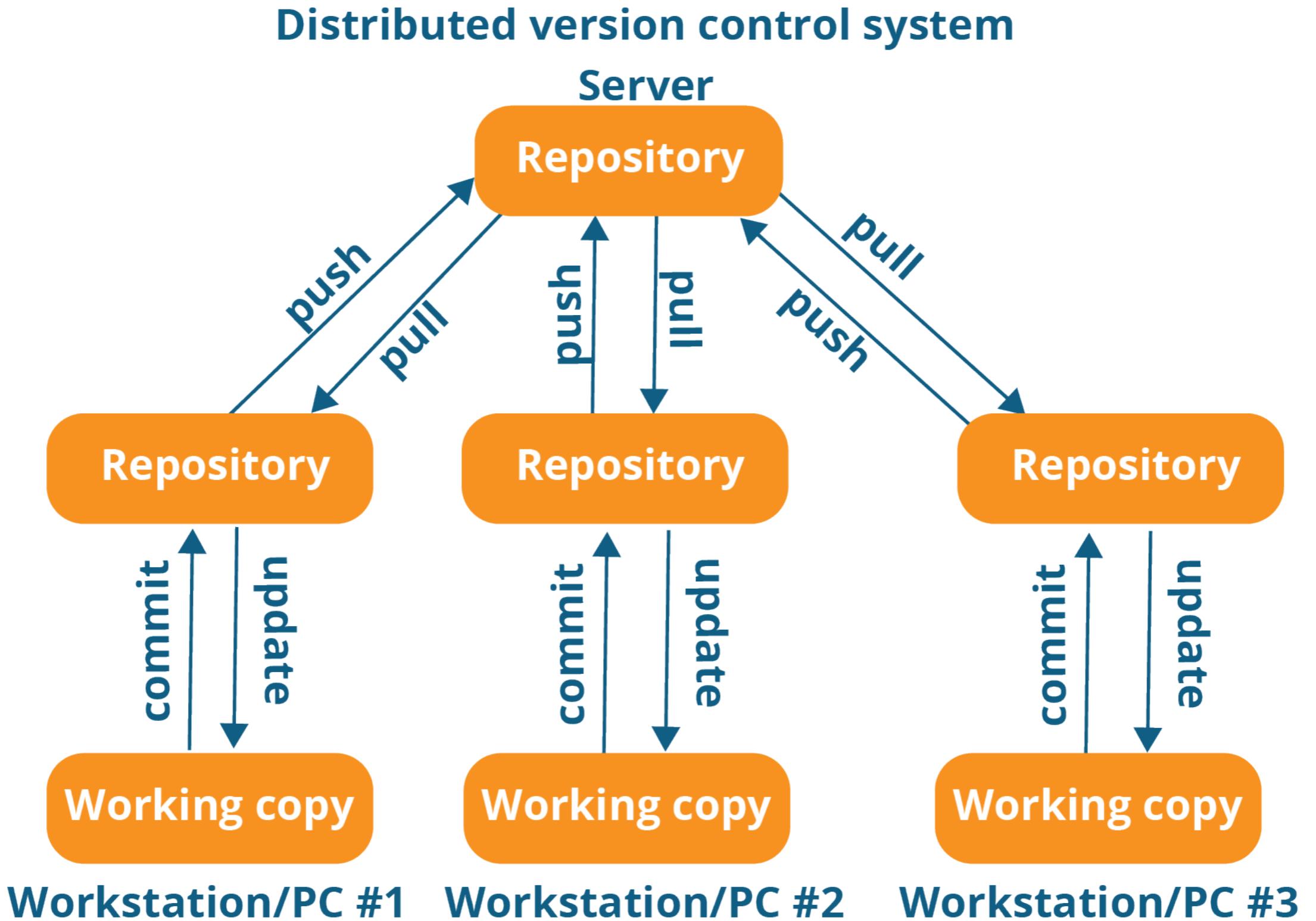
- remote: online
- repository
- checkout
- fork
- branch
- origin
- HEAD
- push
- pull
- Pull Request (PR)
- stash

# Let's get started

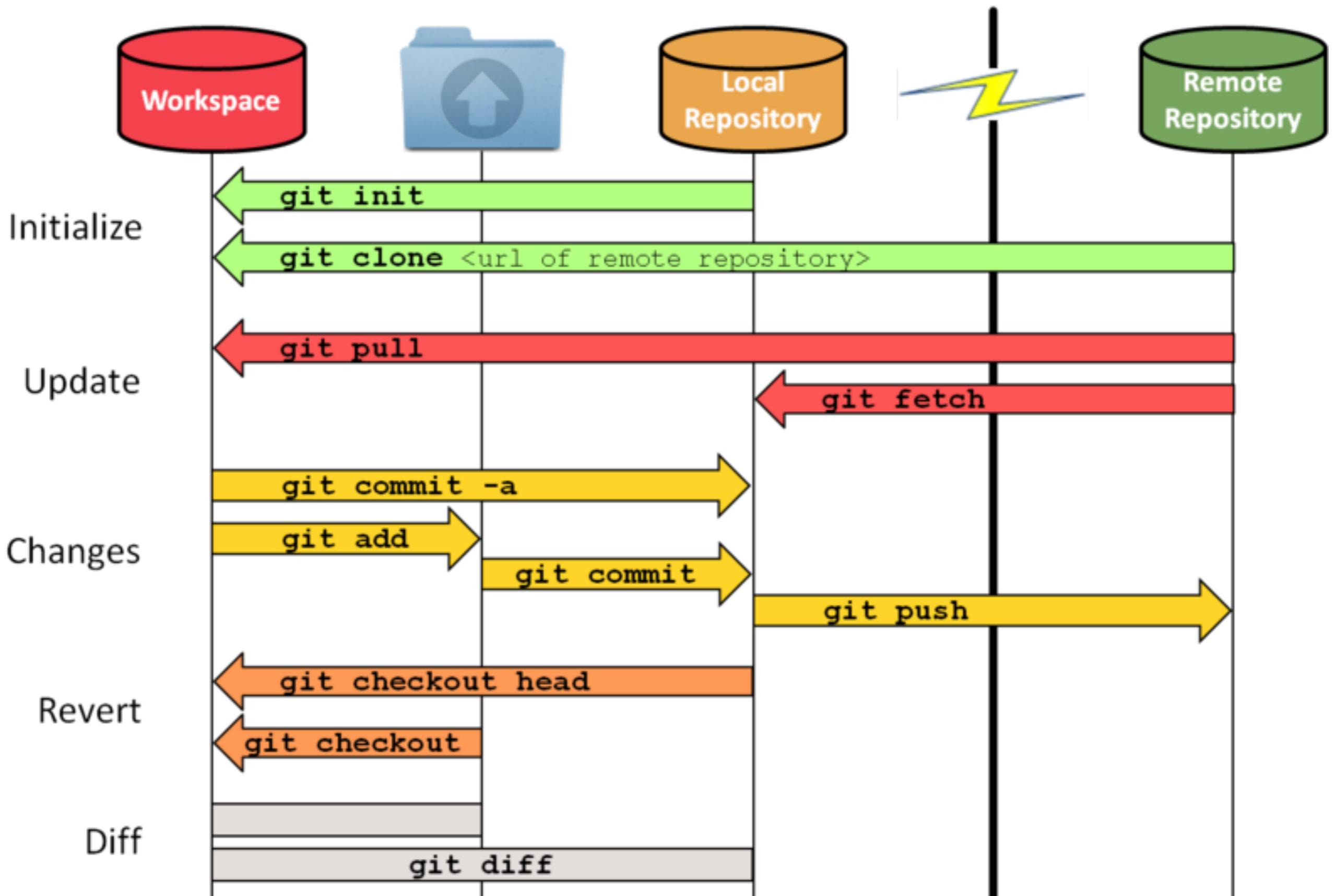
- Git is available on lxplus, HPG, etc.
- In your terminal, go to some work area (~public/ or .../CMSSW\_X)
  - ▶ [/afs.cern.ch/work/d/drosenzw/HiggsMassMeasurement/](https://afs.cern.ch/work/d/drosenzw/HiggsMassMeasurement/) **Show how to fork**
- Now your turn to fork the repo
  - ▶ [https://github.com/rosedjl/CMS\\_Office\\_Hours](https://github.com/rosedjl/CMS_Office_Hours) `git clone https://...`
- Then clone that remote repo to your **local area**
  - ▶ This is your local repo! `git checkout -b mynewbranch`
- Make (checkout) a new branch
  - ▶ Don't work the master branch!
- Begin creating new code!

# Questions to answer:

- Does fetching create a new commit?
- Why clone into commit local repo?



# Git Workflow: Common Commands



# What is Git?

- Git is a software that **tracks your files over time**

- ▶ Changes
  - ▶ History

- This is *version control!*

**What is Git?**

-

- There are many hosts:

- ▶ GitHub
  - ▶ BitBucket
  - ▶ GitLab
  - ▶ gogs
  - ▶ gitea

**Why use Git?**

**- Save your code**

- Git is available on lxplus, HPG, etc.

**Where can I find Git?**

**- All**

**When can we learn Git?**  
***Right now!***