

CS 6015: Software Engineering

Spring 2024

Lecture 3: Version Control

Last Week

- C++ classes / declaration and implementation
- C++ header files ? Misc code snippet
- Using command line (Shell) to compile
- Makefiles

This Week

- Version/source control
- Testing
- Code review today
- Assignment 2 released – due next Tuesday

Working on a project alone

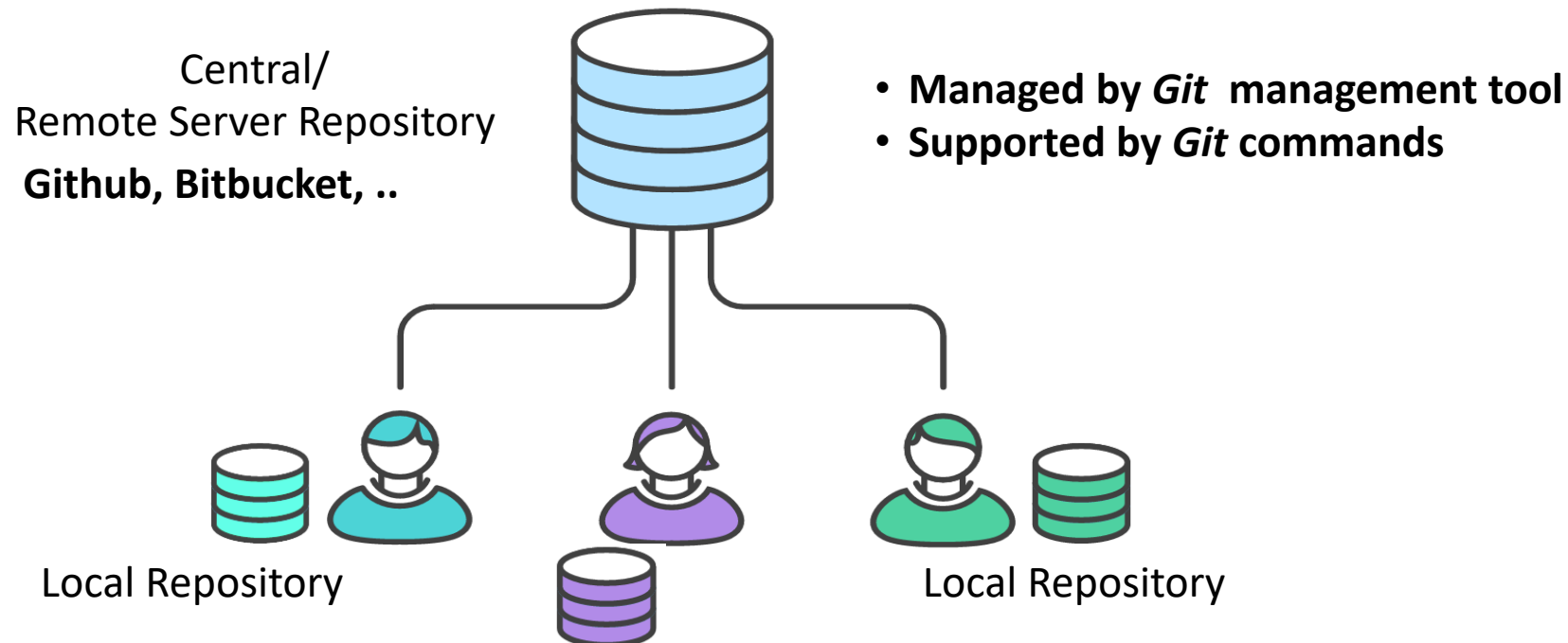
- How to save your data?
 - External HDD
 - Dropbox
- What problems would you face?
 - Different saved versions
 - Early and frequent savings
 - Conflicts
- Program breaks and need to restore your code/data
 - Which is the latest copy?
 - What happens if some intermediate version of the project was not copied

Working on a project in a team

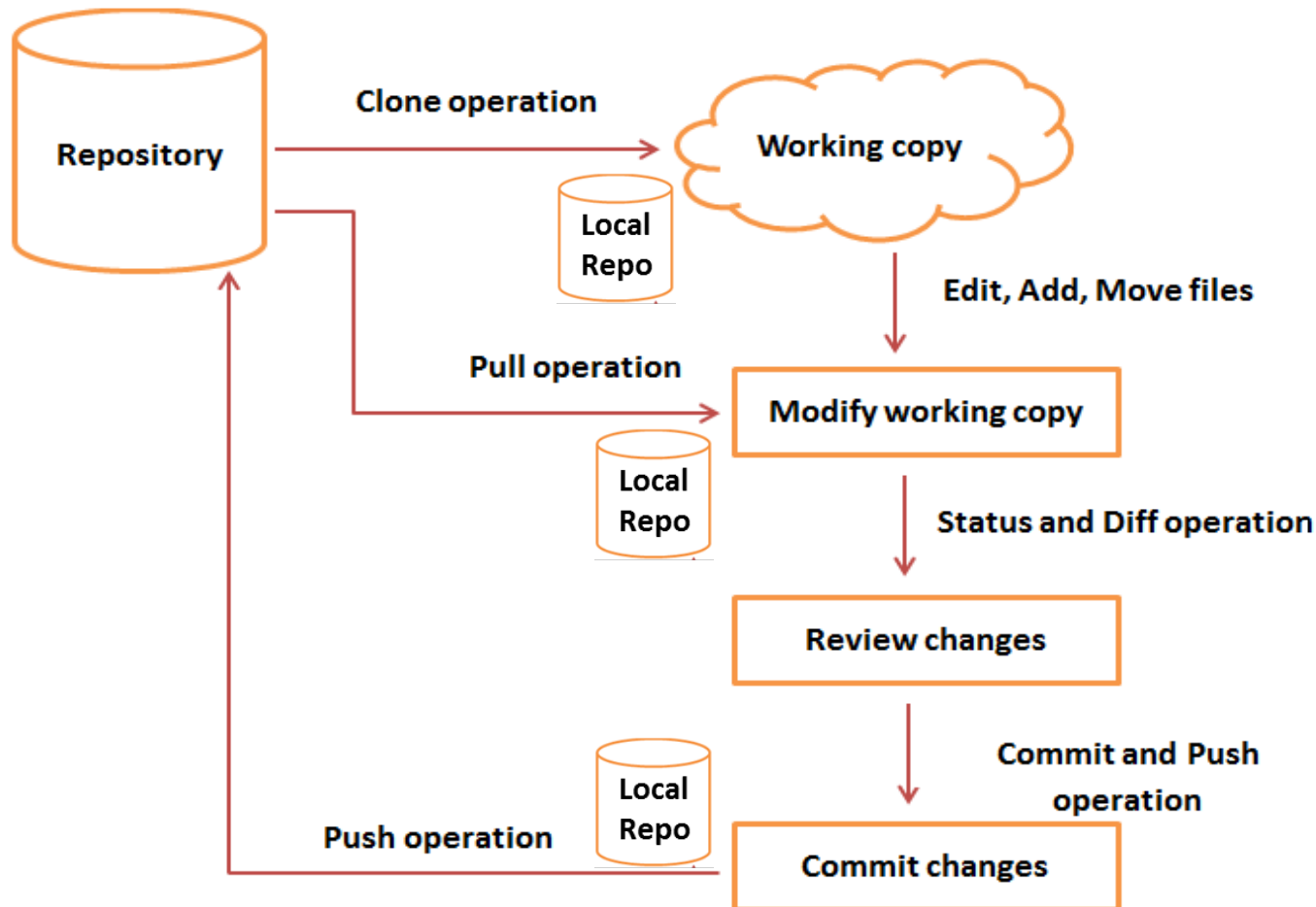
- How to save your data?
 - Shared folder
 - Exchange code via email/slack
- How would you need to handle?
 - Maintaining different versions on the shared folder.
 - Synching changes.
- Solution
 - Version control (Source control)

Version control

- Main components
 - Git: **distributed** version control system for source code management
 - Github: web-based hosting service to store the data remotely



Git operations: Globally



- ***git clone <url>***: creates a local copy of the central repository
-

Before pushing new changes:

- ***git add***: add changes in the working directory to the staging area.
- ***git commit -m "msg"***: Commit the staged snapshot to the project history

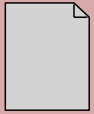
Then:

- ***git push***: apply changes in local repo to central repo
-

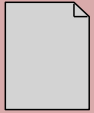
- ***git pull***: apply changes in central repo to local repo

Git operations – locally

Working folder



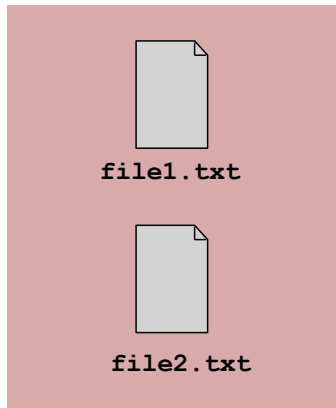
`file1.txt`



`file2.txt`

Git operations – locally

Working area



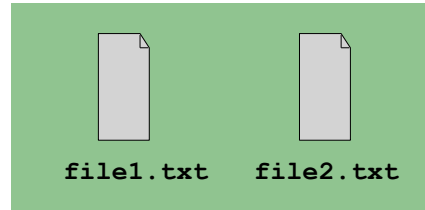
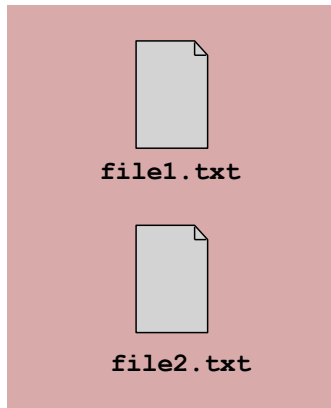
Staging area

Local Repository



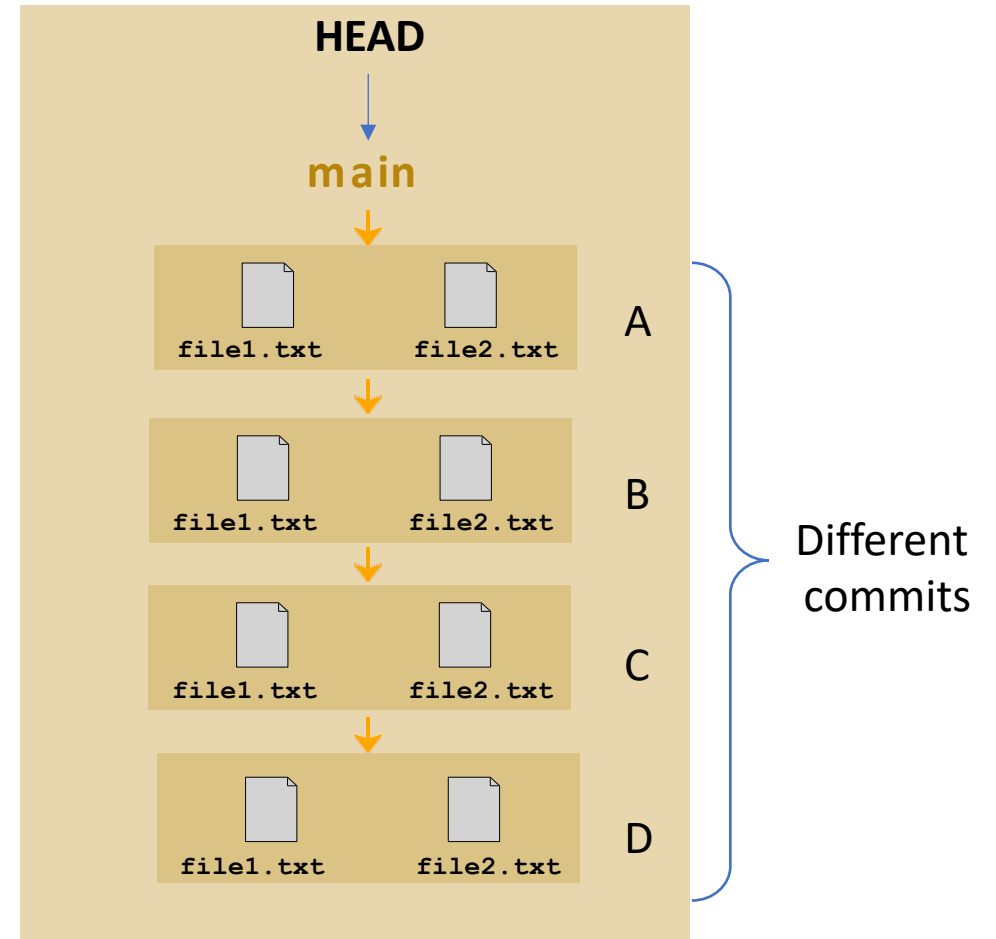
Git operations – locally

Working area



Staging area

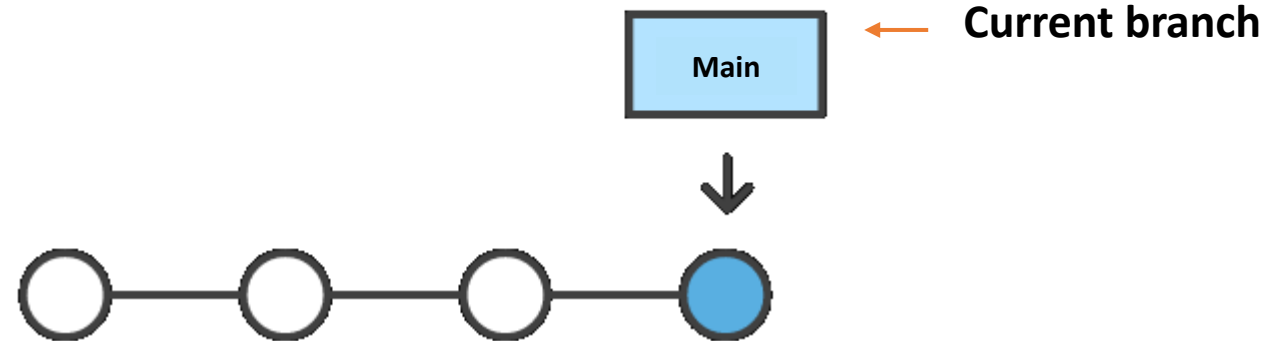
Local Repository



Git

- Consider a supermarket program
- How to add some new features?

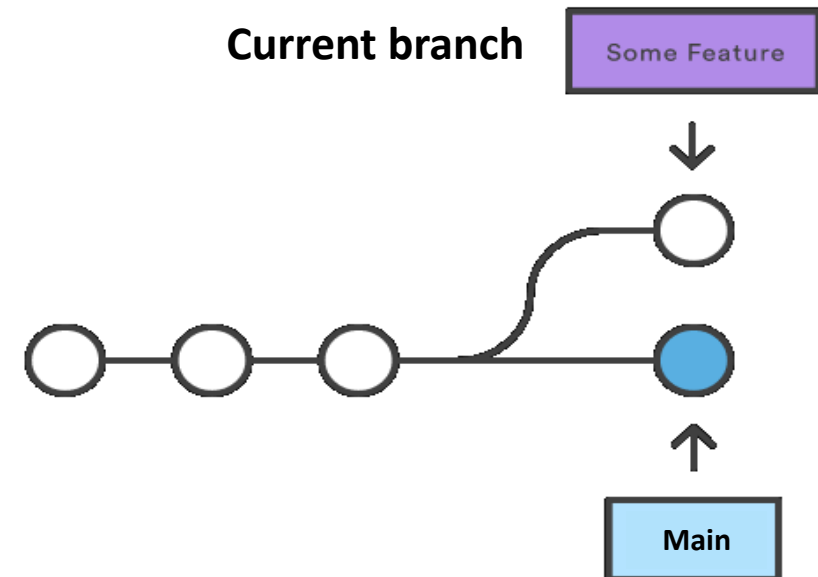
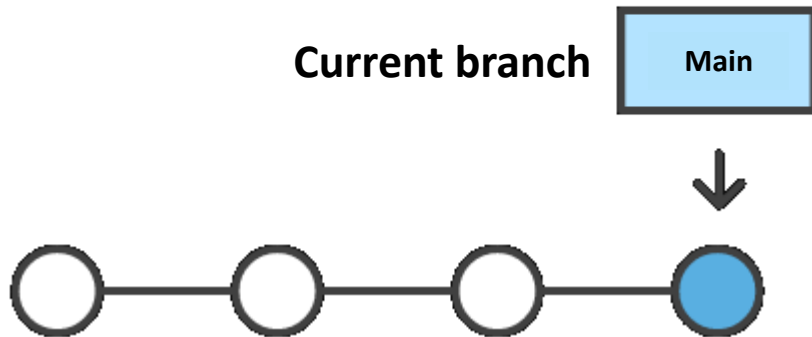
- Check sold items
- Check duplicates
- Find total amount



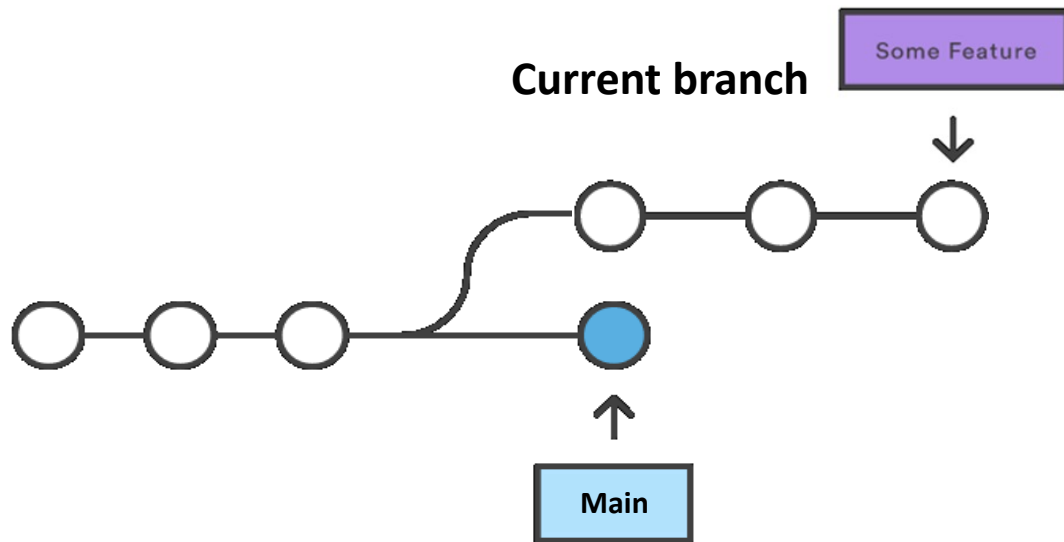
- Concern
 - Decided not to go with the new features at a later stage
 - Do NOT want to change the files for the working program

Git: branches

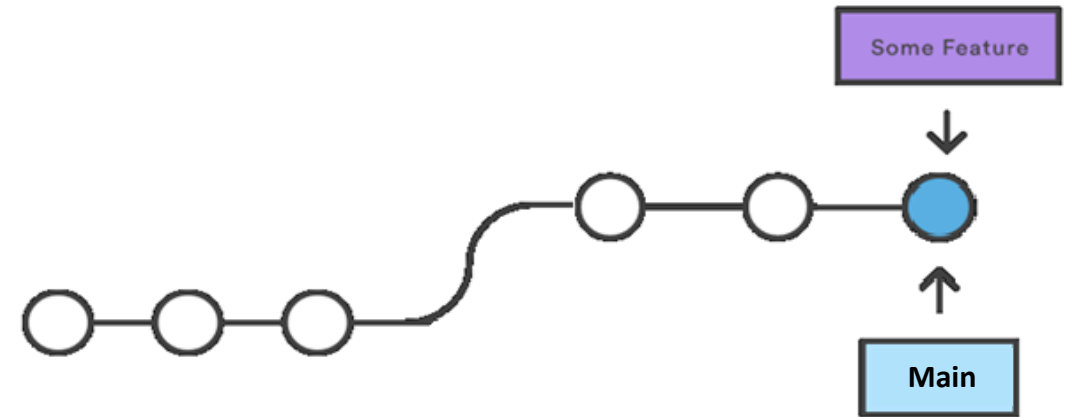
- Solution: use branches
- Steps:
 - Create branch: ***git branch <name>***
 - Move to branch: ***git checkout <name>***



Git: branches



Add and commit changes to the branch

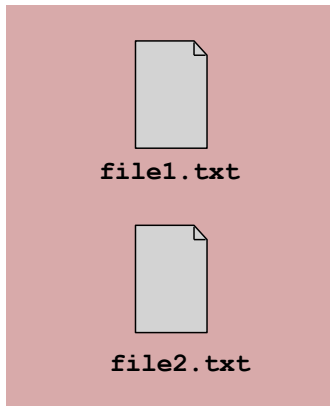


Once done, move back and merge to master:

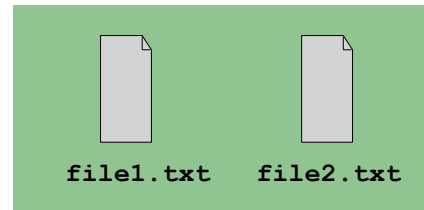
- ***git checkout main***
- ***git merge <branch>***

Git operations – locally

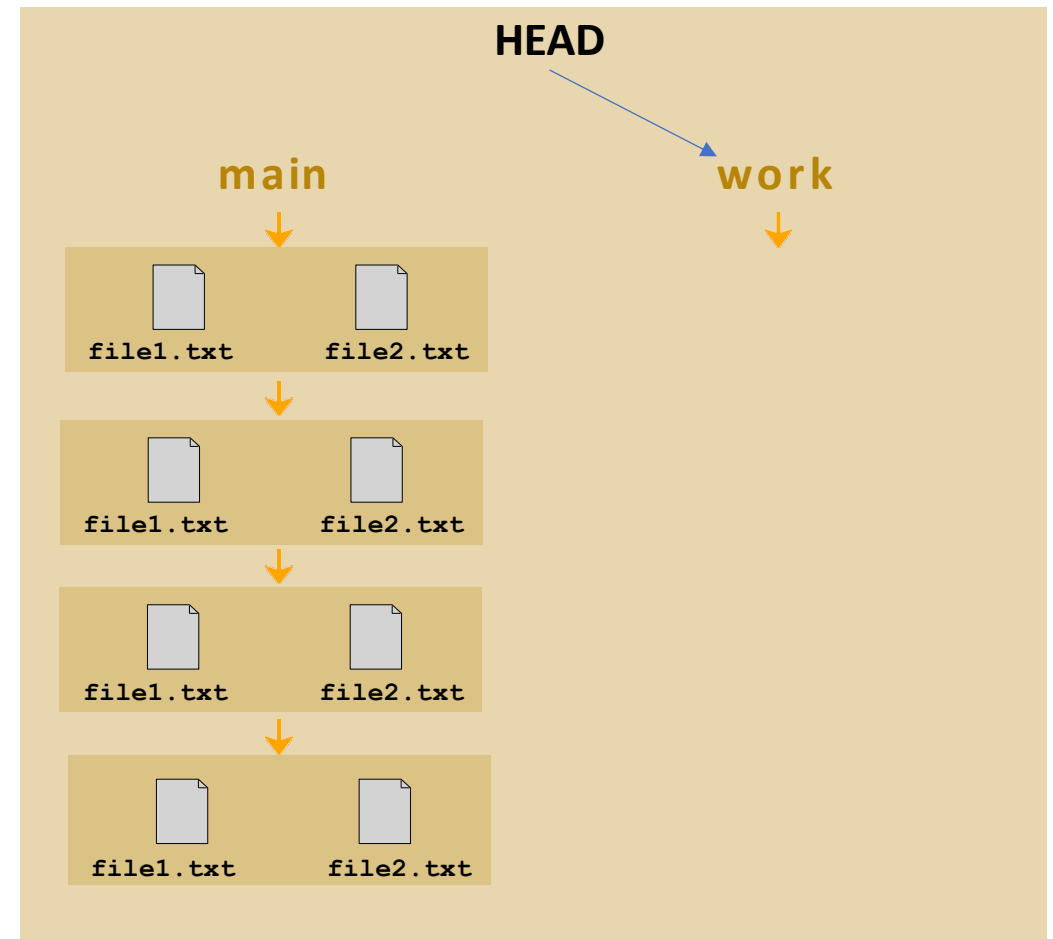
Working area



Staging area



Local Repository



Git: Sketching full example

- The **main** branch of your Git repository is for **communicating** code
- The **main** branch is not a **backup** mechanism, **don't** have a history in **main** that looks like this:

```
commit a045f9 first cut
commit b788cd part way there
commit 9345ab I was confused
commit cd7723 most tests now pass
...
```

- A commit on the **main** branch should generally be a working version
- But you should *absolutely* back up your work along the way, and a branch other than **main** is a fine way to backup work

Git: Sketching full example

- Create a **work** branch:

```
$ git branch -d work // Deletes existing work branch  
$ git branch work // Creates new branch work  
$ git checkout work // Switches HEAD to work branch
```

Git: Sketching full example

- Create a **work** branch:

```
$ git branch -d work  
$ git branch work  
$ git checkout work
```

- Make changes and periodically save work

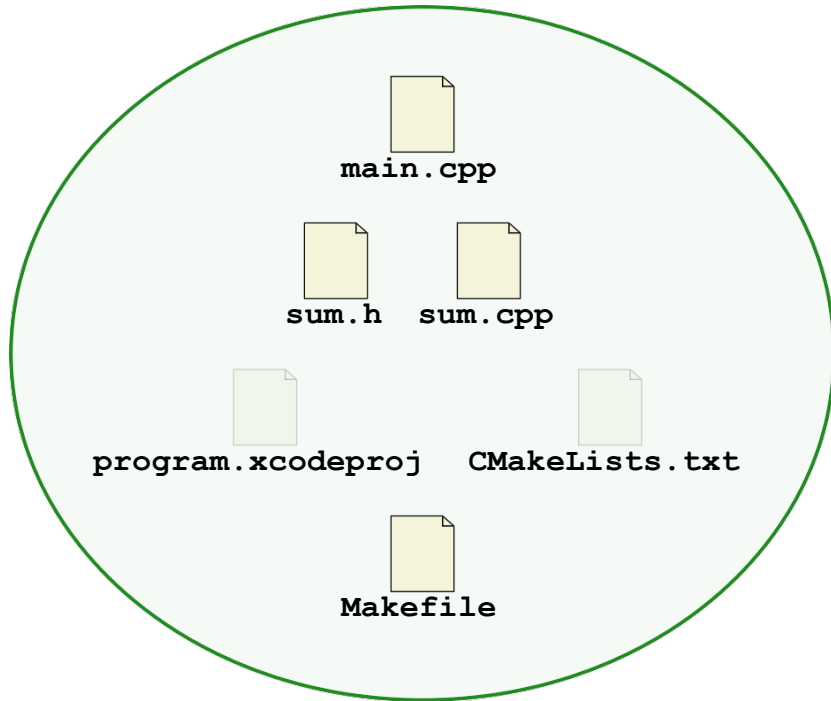
```
$ git add .  
$ git commit -m "whatever... work in progress"  
$ git push origin work
```

- At a working state, switch back to

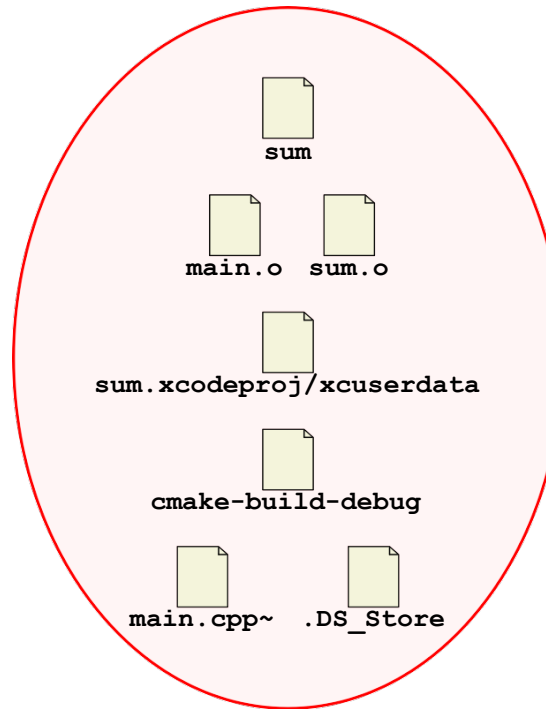
```
$ git checkout main  
$ git merge --squash work // Stages all work changes in main  
$ git commit -m "nice description for others to read"
```


Git: Sketching full example

Keep in repo



Exclude via `.gitignore`



Exclude via ***.gitignore***:

- Compiled executables
- Build intermediate
- IDE ephemeral state
- Backup files
- Finder layout

Git and github: Sketching full example

- At **github.com**, click the **+** in the top right and select **New repository**
- Since it's for homework, make the repository **Private**.
- Your account should be the owner.
- On your machine, make a directory for your repository
- **cd** there, put files there including **.gitignore**, and use

Assuming local repo exists

```
$ git init
$ git add .
$ git commit -m "initial version"
$
```

```
$ git remote add origin git@github.com:user/repo
$ git push -f -u origin main
```

New name of the remote repo

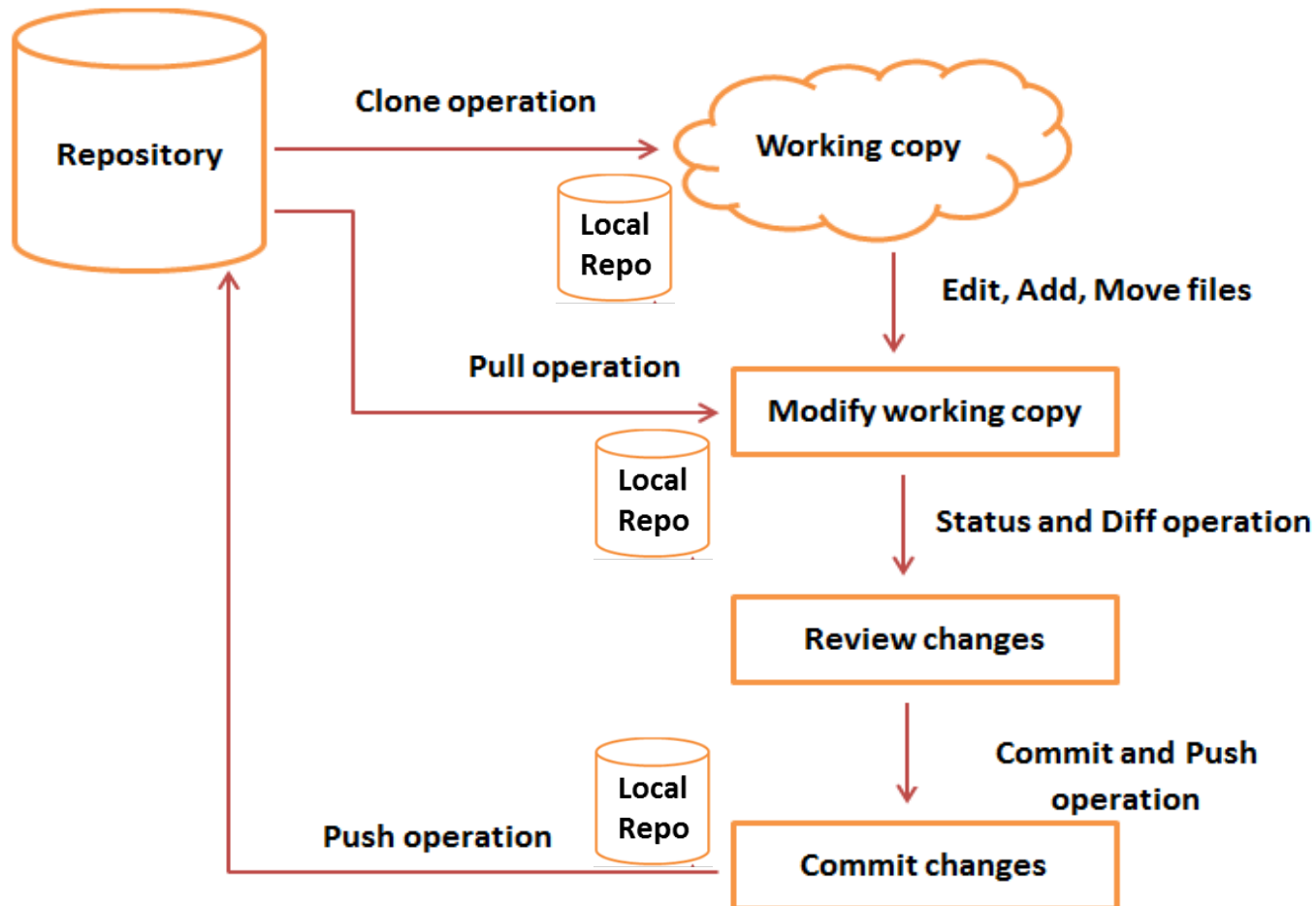
Or: after creating the repo
online, git clone ...

Git and github: Sketching full example

- At **github.com**, click the **+** in the top right and select **New repository**
- Since it's for homework, make the repository **Private**.
- Your account should be the owner.
- On your machine, make a directory for your repository
- **cd** there, put files there including **.gitignore**, and use

```
$ git init
$ git add .
$ git commit -m "initial version"
$ git branch -M main
$ git remote add origin git@github.com:user/repo
$ git push -u origin main
```

Git operations



- ***git clone <url>***: creates a local copy of the central repository
-

Before pushing new changes:

- ***git add***: add changes in the working directory to the staging area.
- ***git commit -m "msg"***: Commit the staged snapshot to the project history

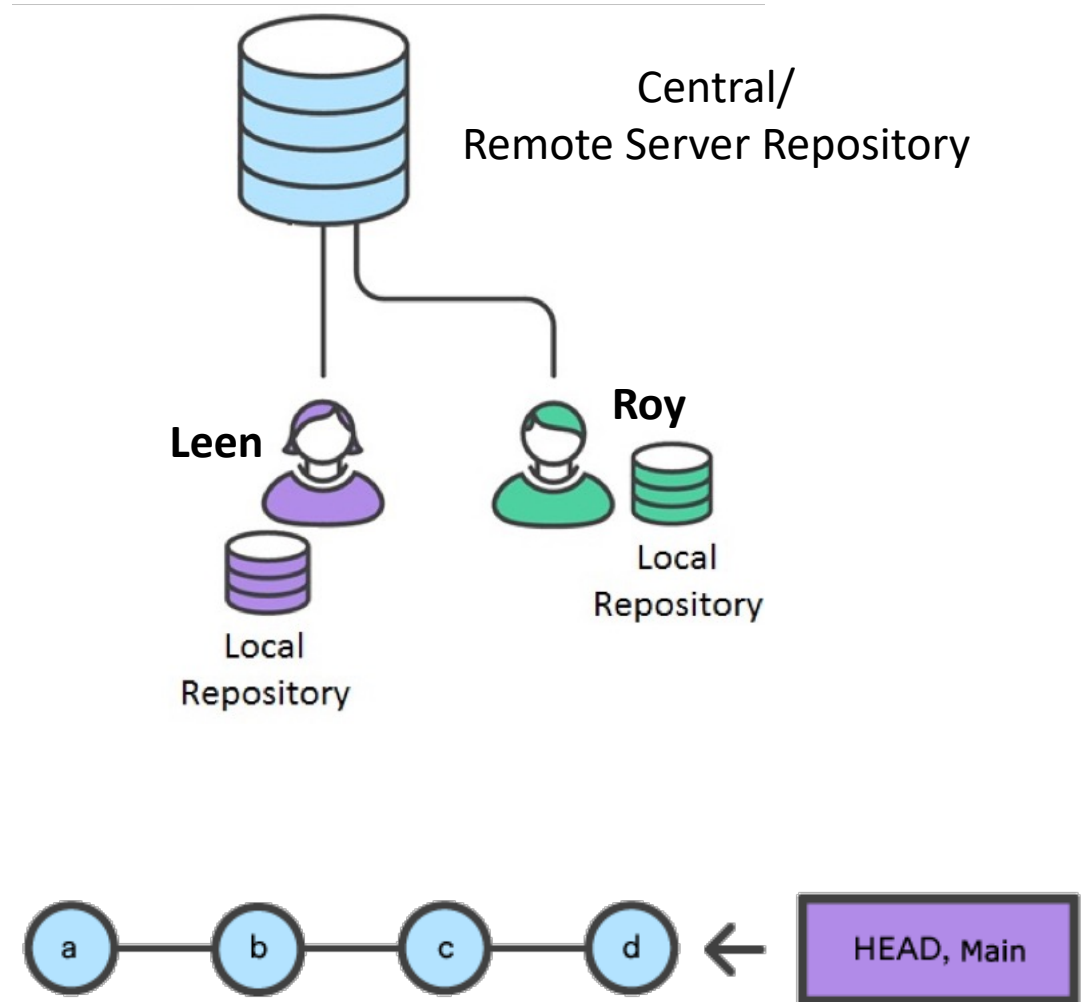
Then:

- ***git push***: apply changes in local repo to central repo
-

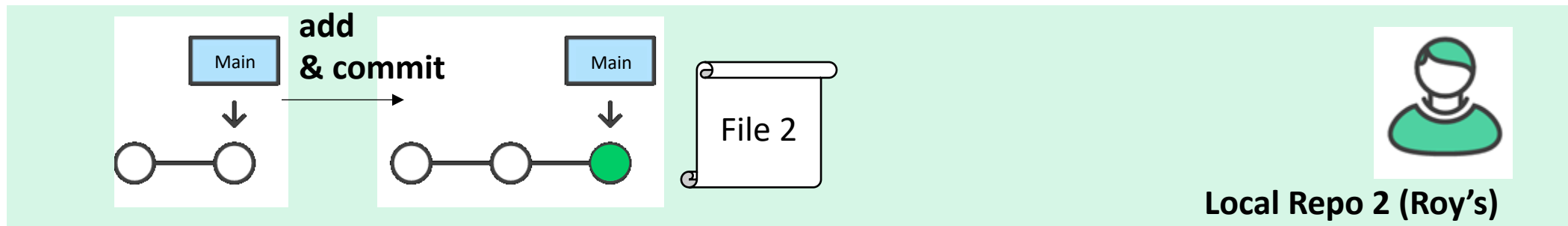
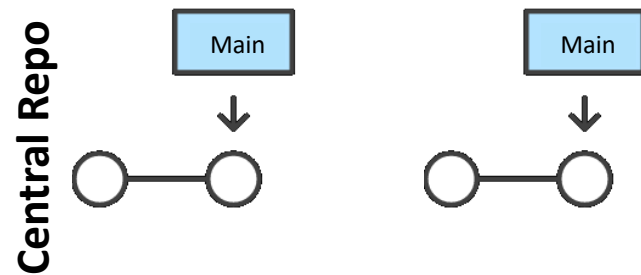
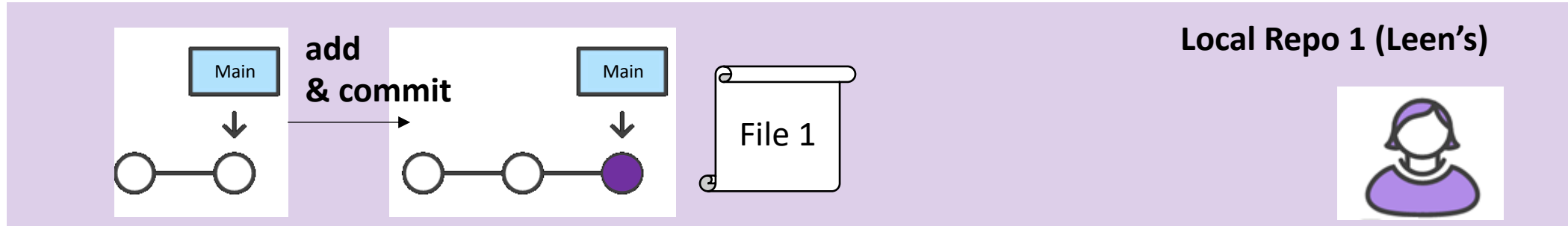
- ***git pull***: apply changes in central repo to local repo

Git and github

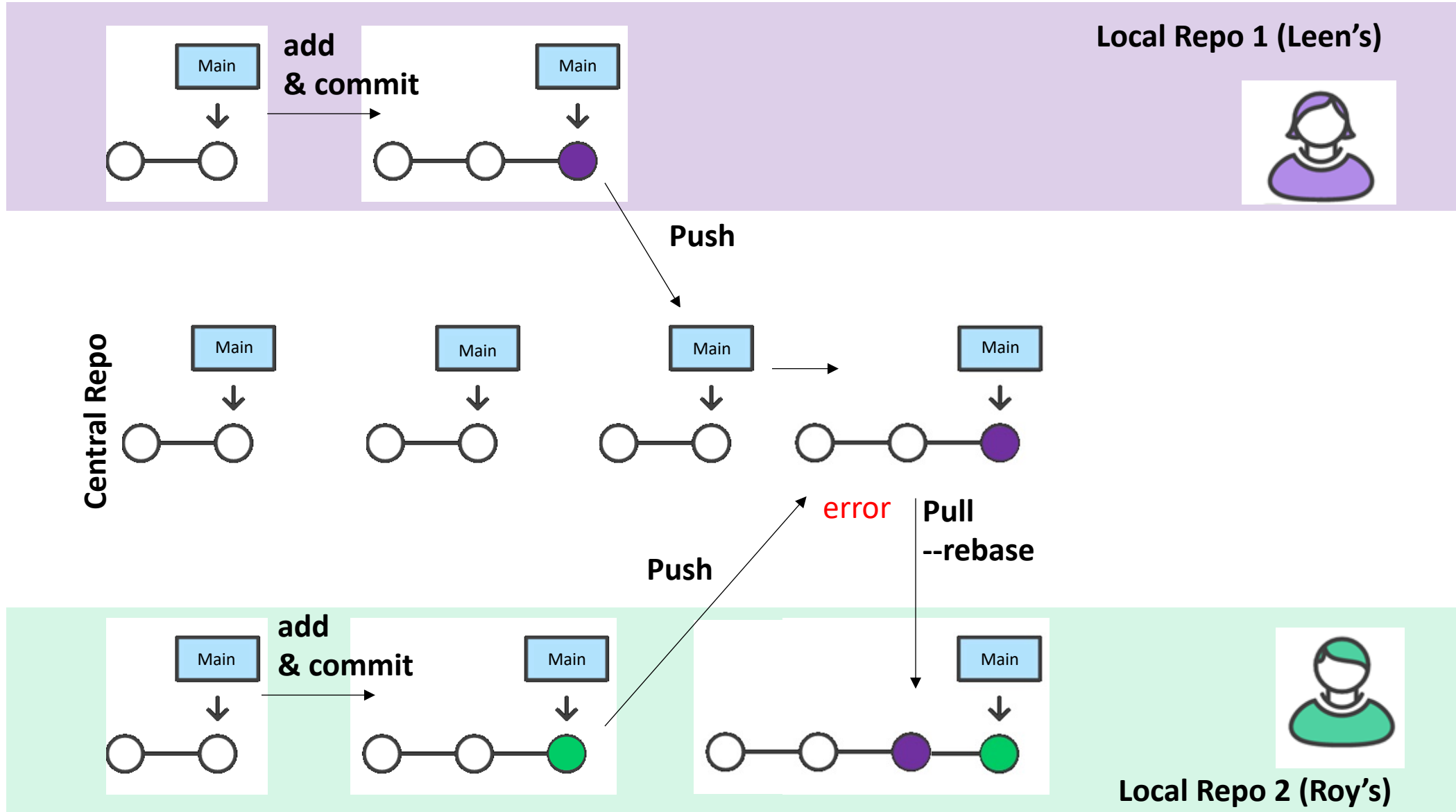
- What could go wrong?
- Scenario 1
- Scenario 2



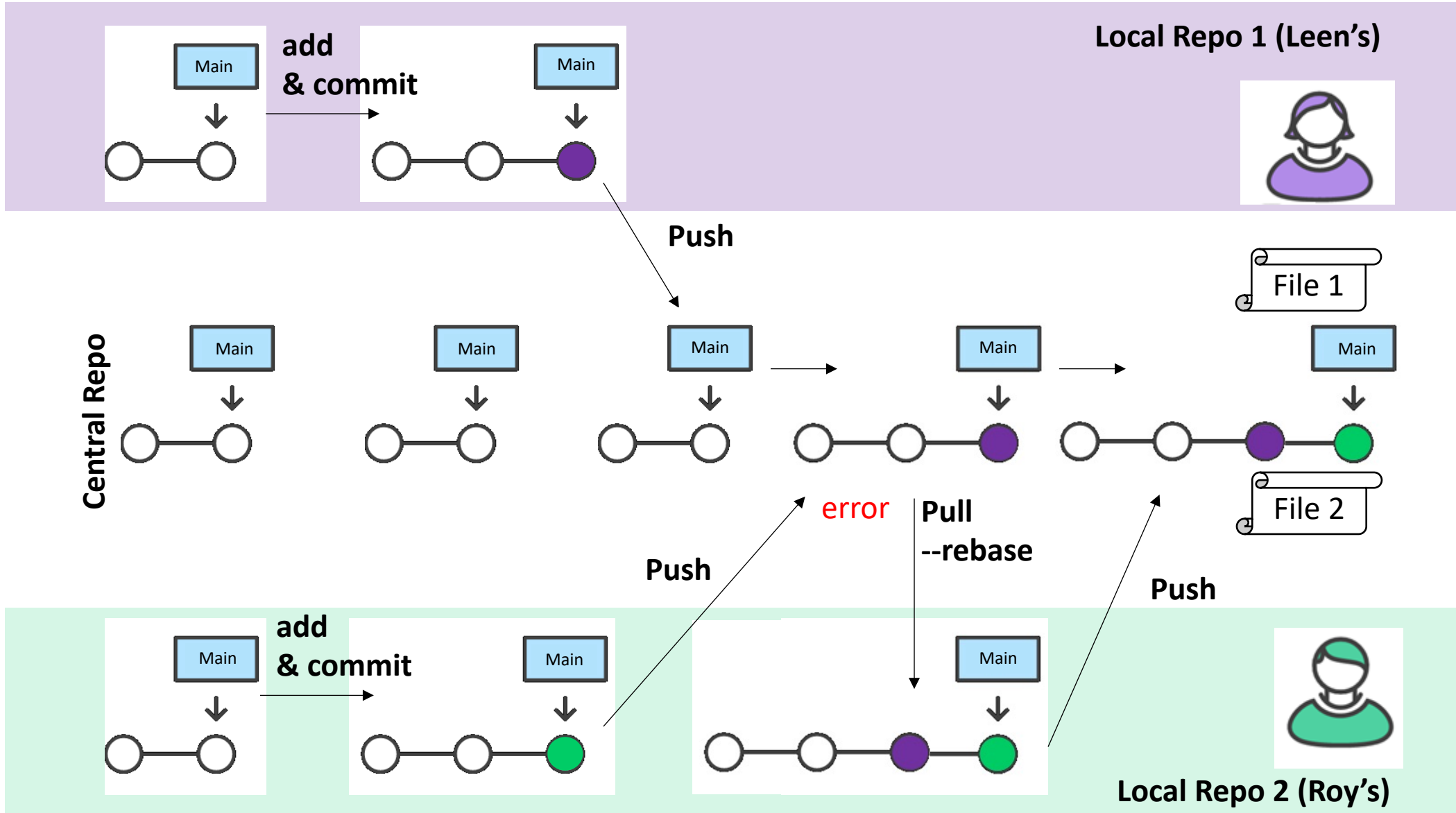
Scenario 1



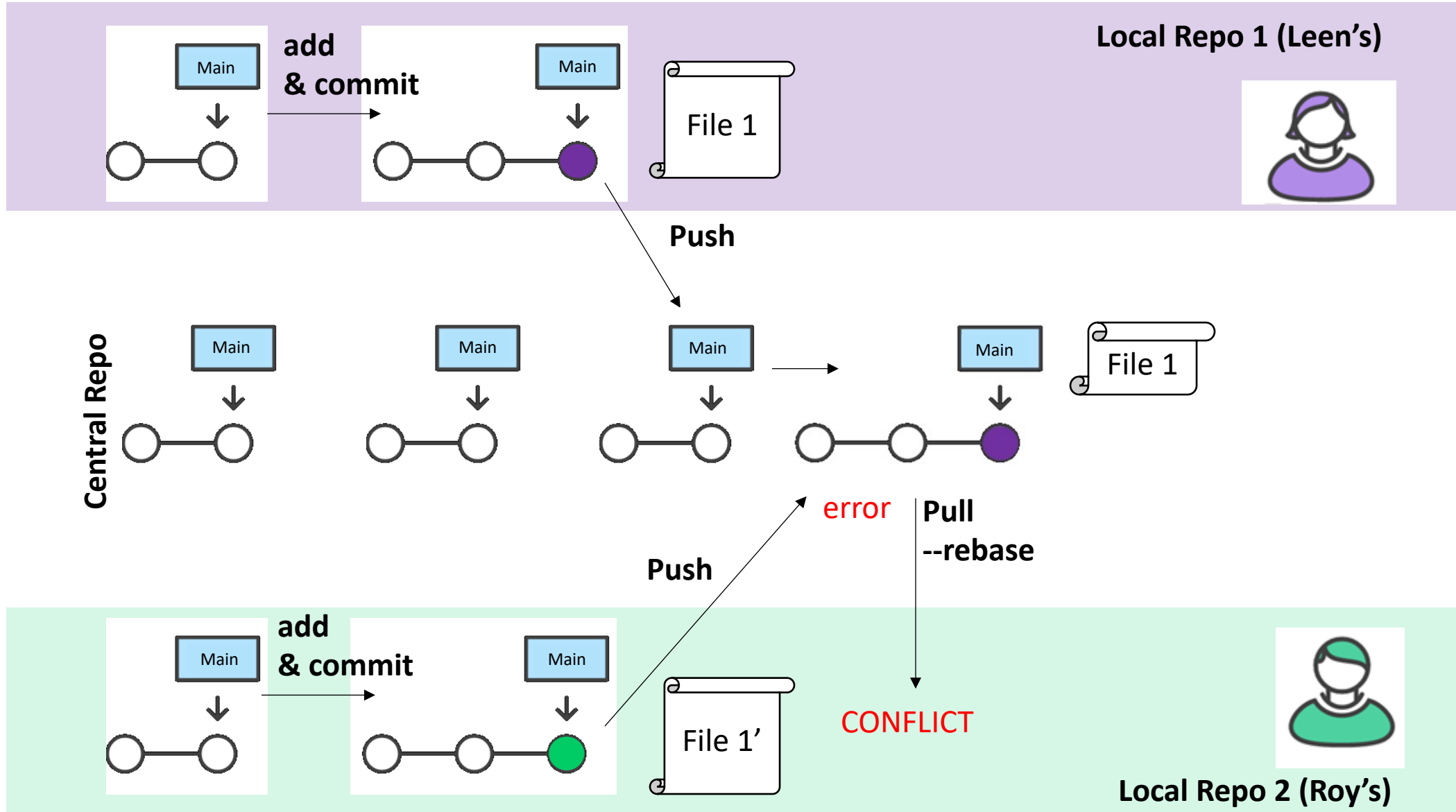
Scenario 1



Scenario 1

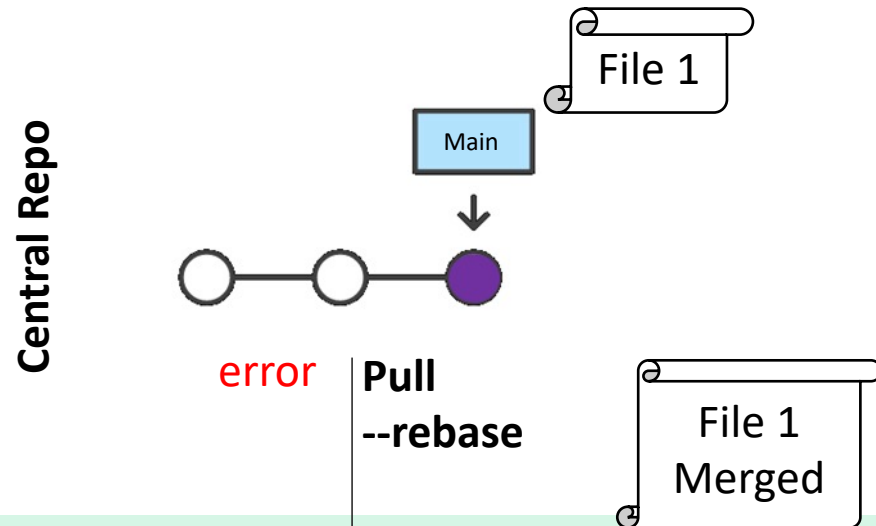


Scenario 2

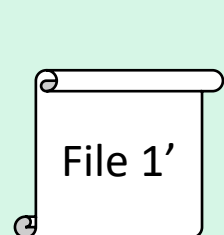


Scenario 2

Local Repo 1 (Leen's)



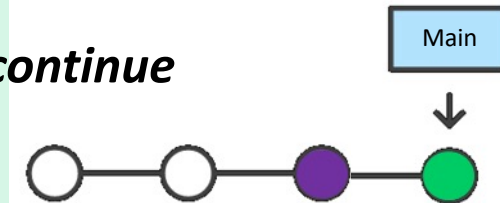
CONFLICT



git status

Edit
conflicting
files

git rebase --continue

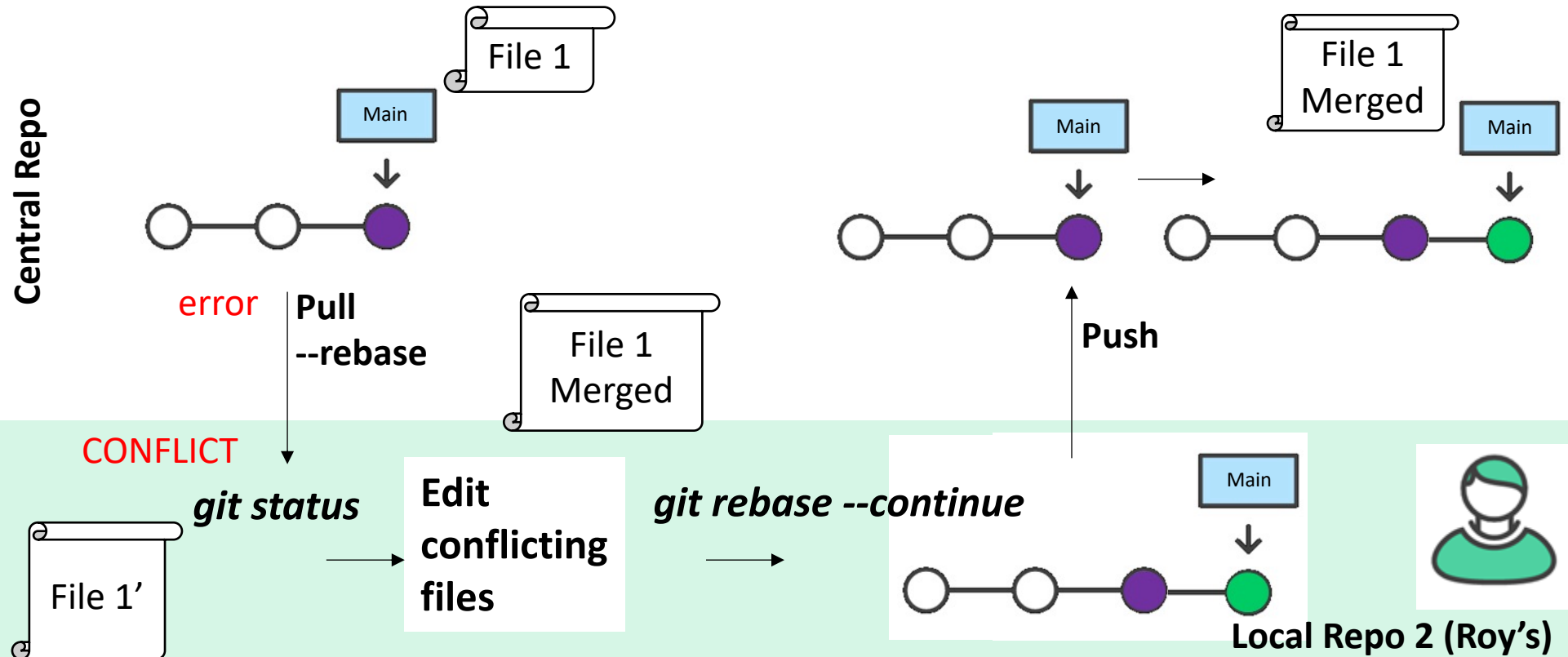


Local Repo 2 (Roy's)



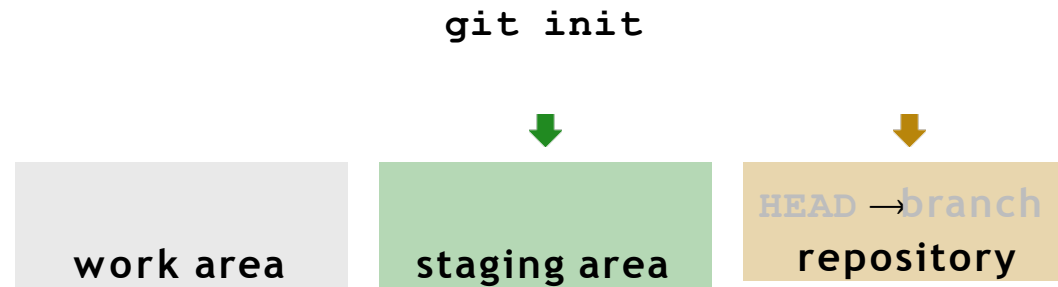
Scenario 2

Local Repo 1 (Leen's)

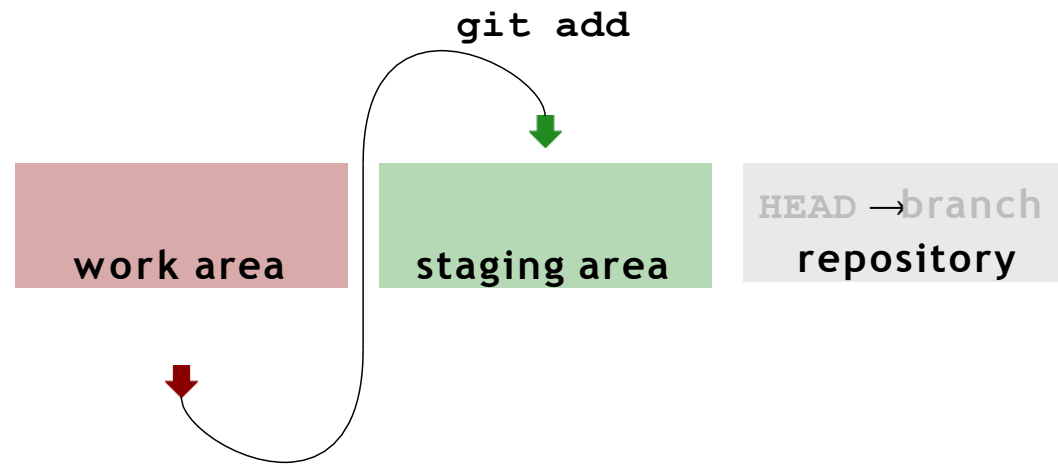


Backup slides: More Git

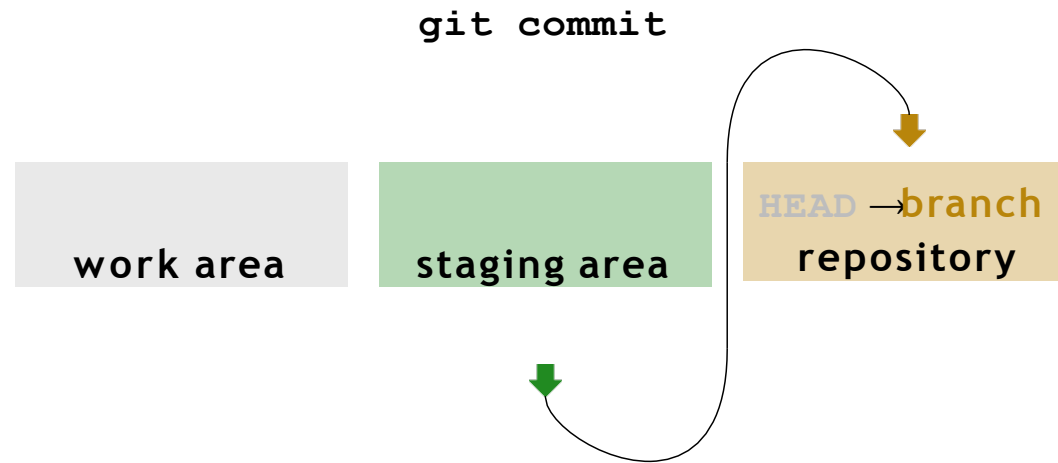
Git Command



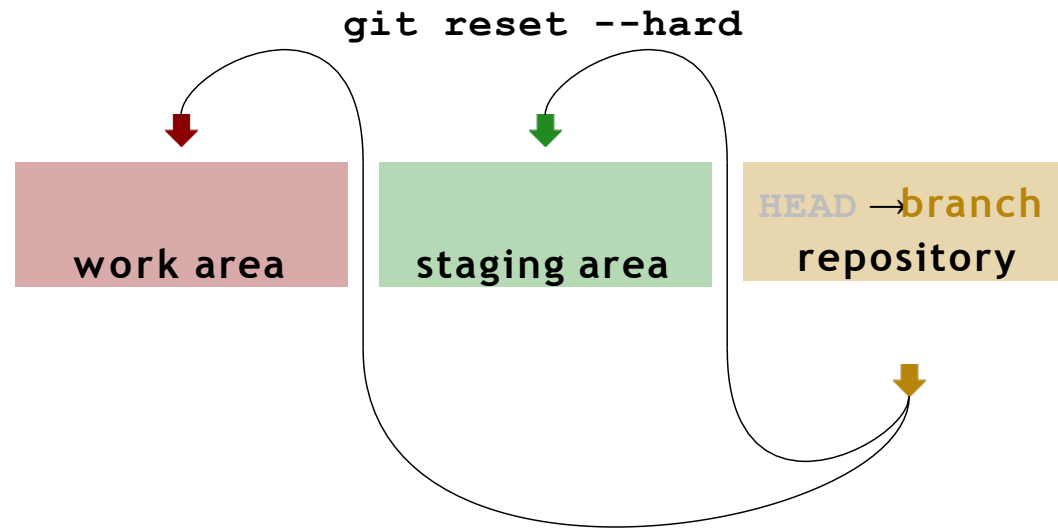
Git Command



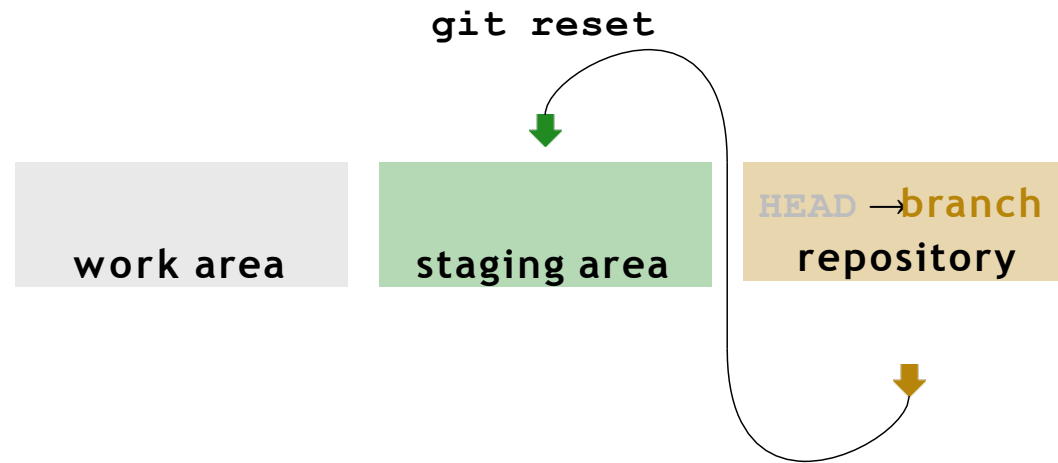
Git Command



Git Command



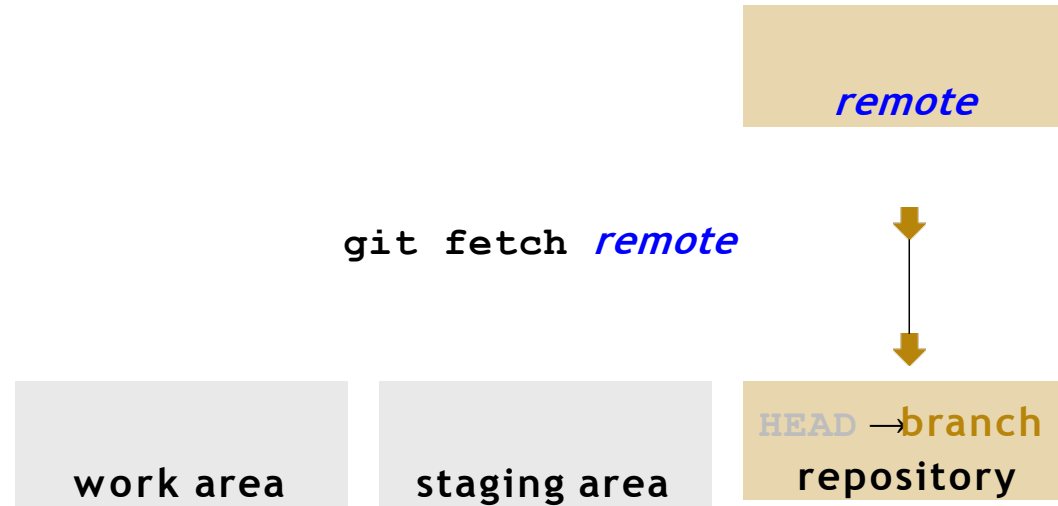
Git Command



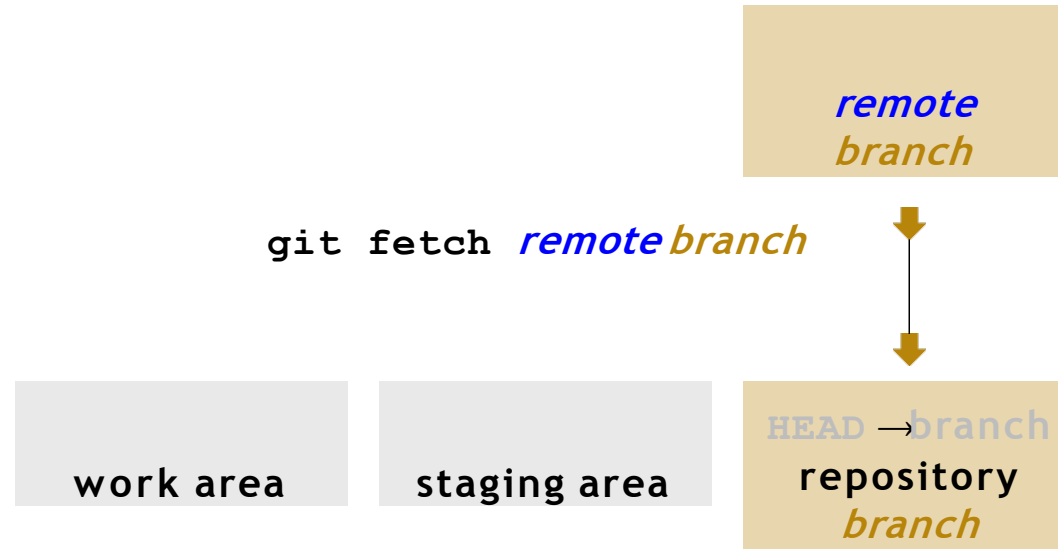
Git Command



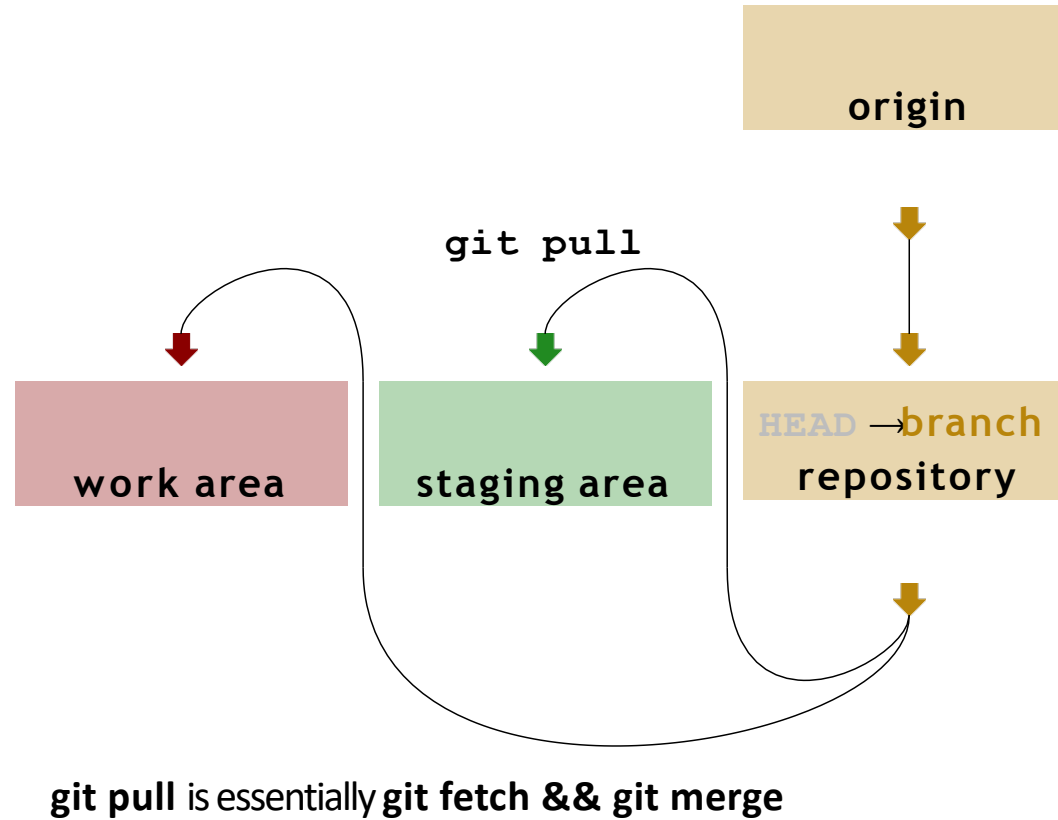
Git Command



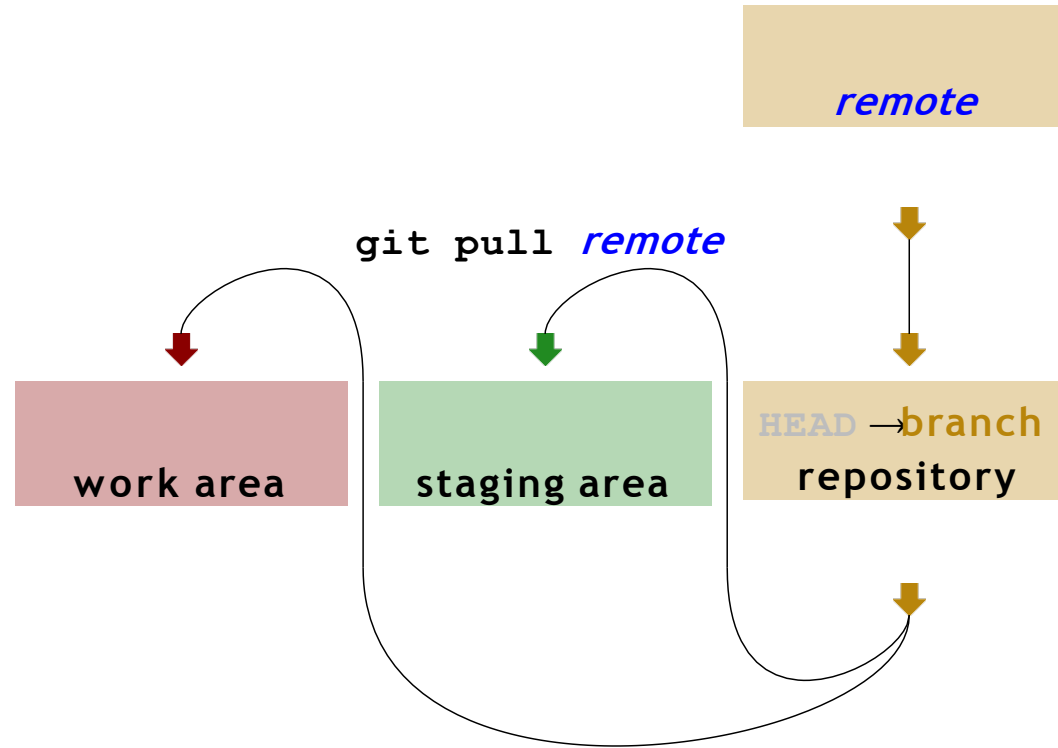
Git Command



Git Command

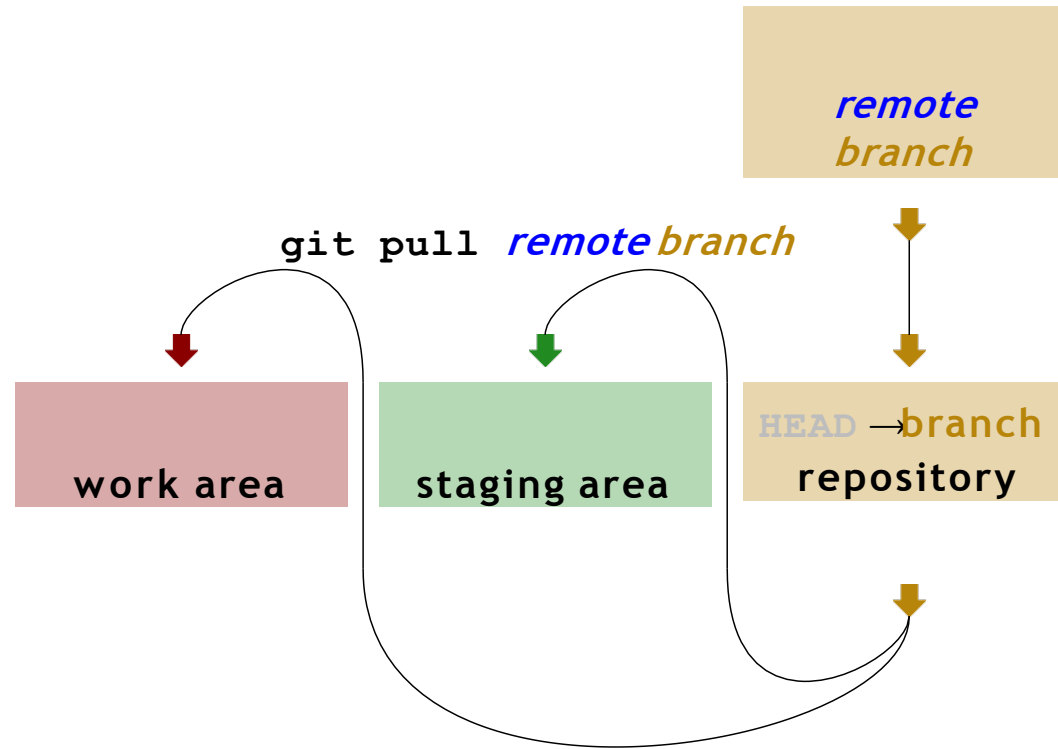


Git Command



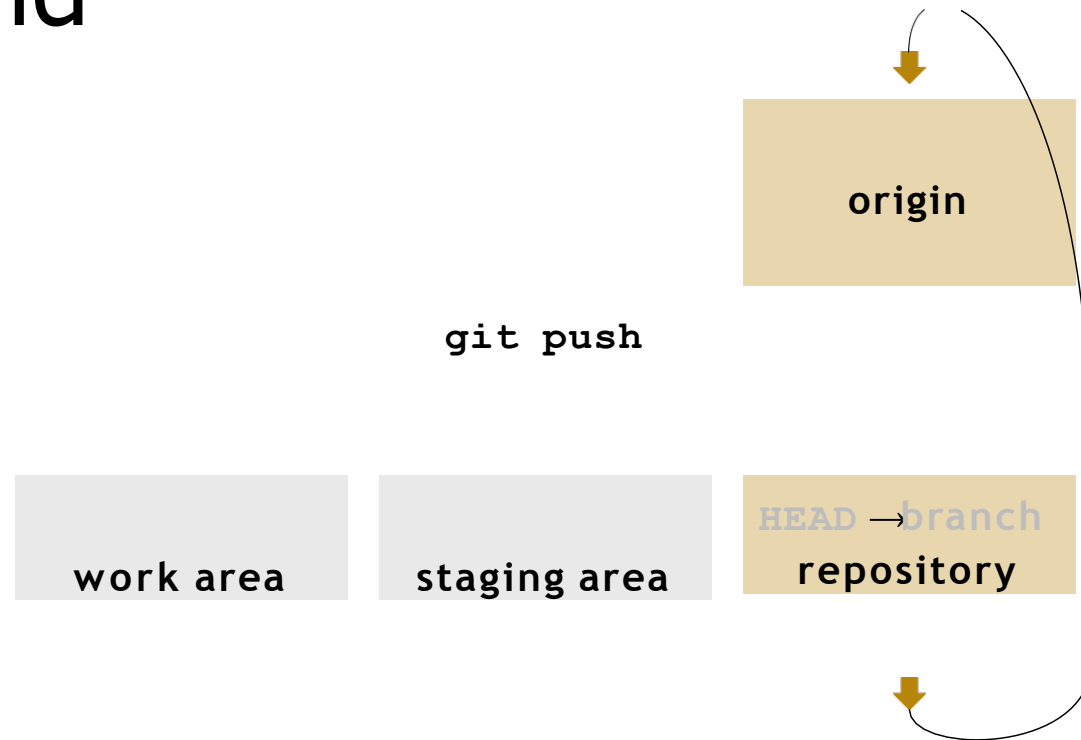
git pull is essentially **git fetch && git merge**

Git Command

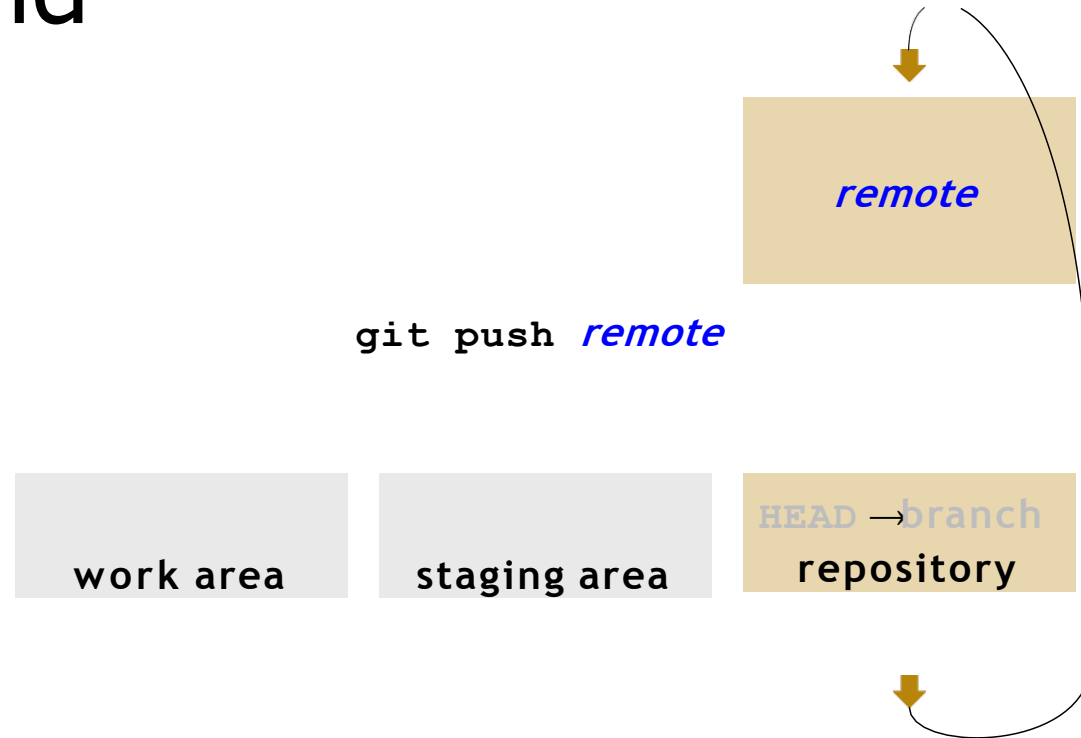


git pull is essentially **git fetch** && **git merge**

Git Command



Git Command



Git Command

