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

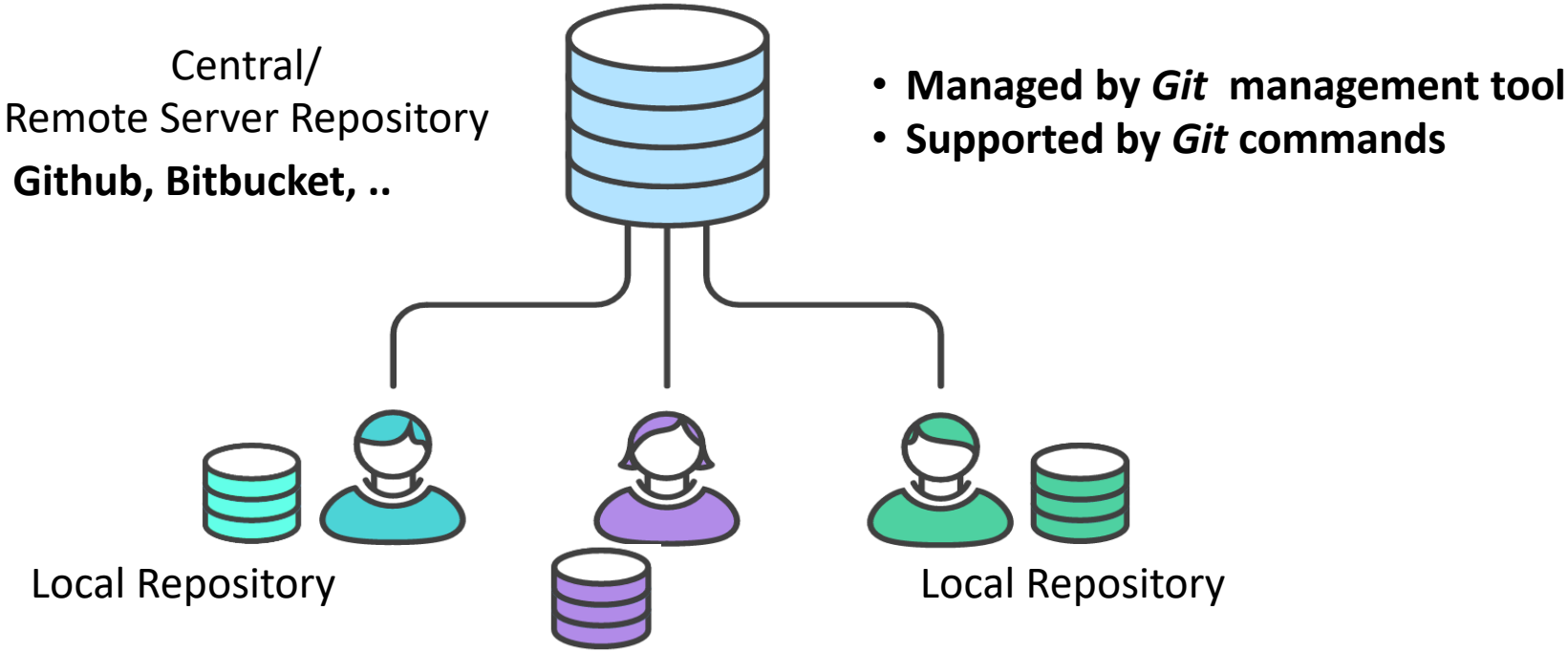
- How to save your data?
  - External HDD
  - Dropbox
- What problems would you face?
  - Different saved versions
  - Early and frequent savings
  - Conflicts
- Program breaks and need to restore your code/data
  - Which is the latest copy?
  - What happens if some intermediate version of the project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

Working on a project in a team

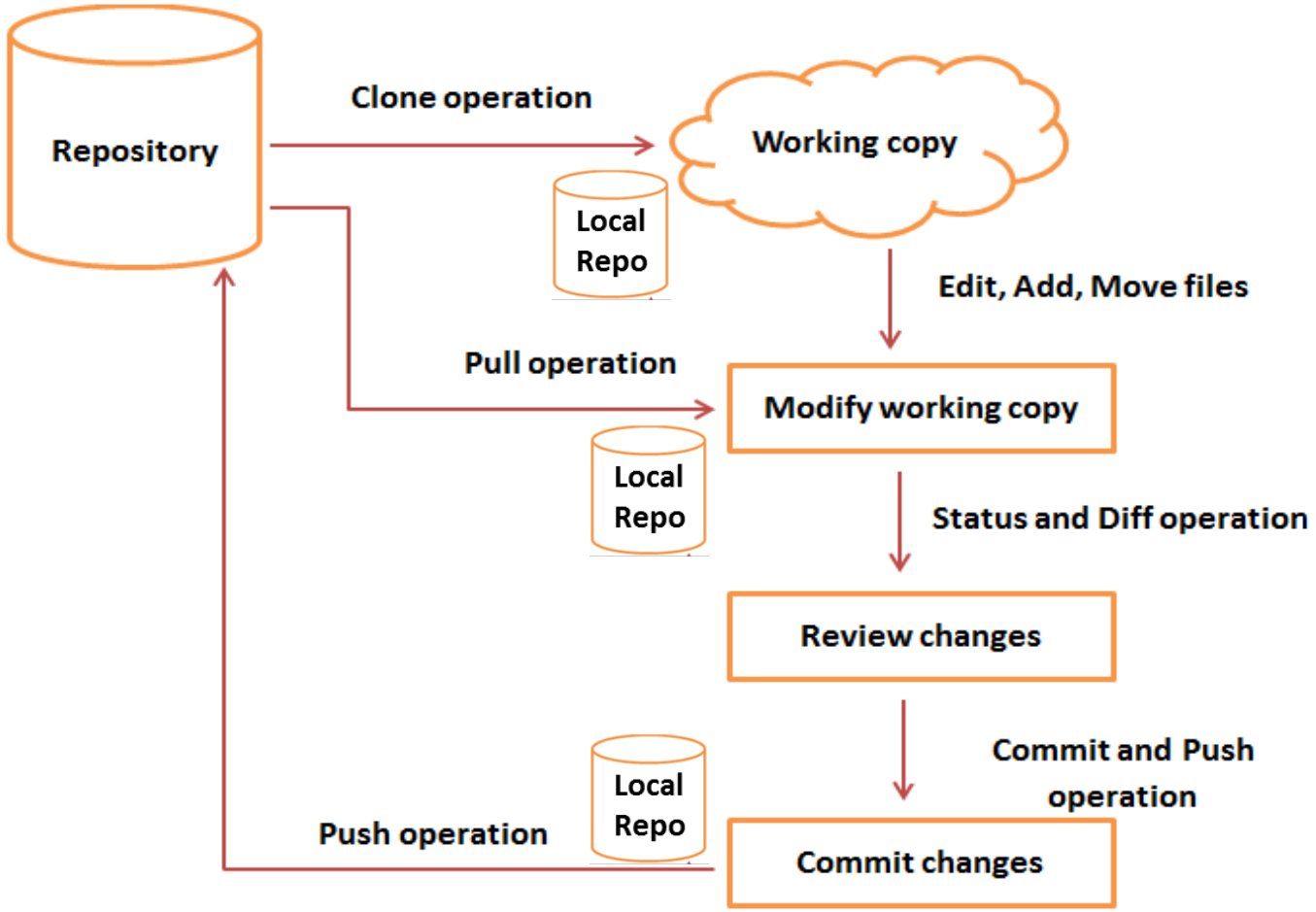
- How to save your data?
  - Shared folder
  - Exchange code via email/slack
- How would you need to handle?
  - Maintaining different versions on the shared folder.
  - Synching changes.
- Solution
  - Version control (Source control)

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

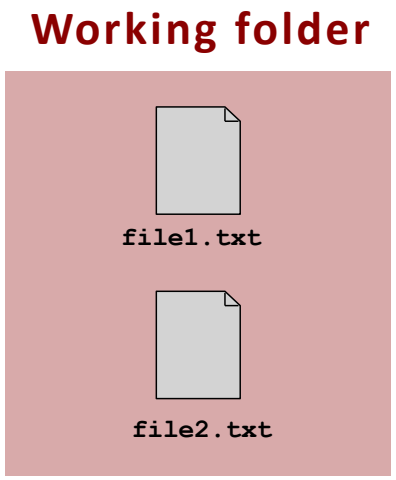
Then:

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

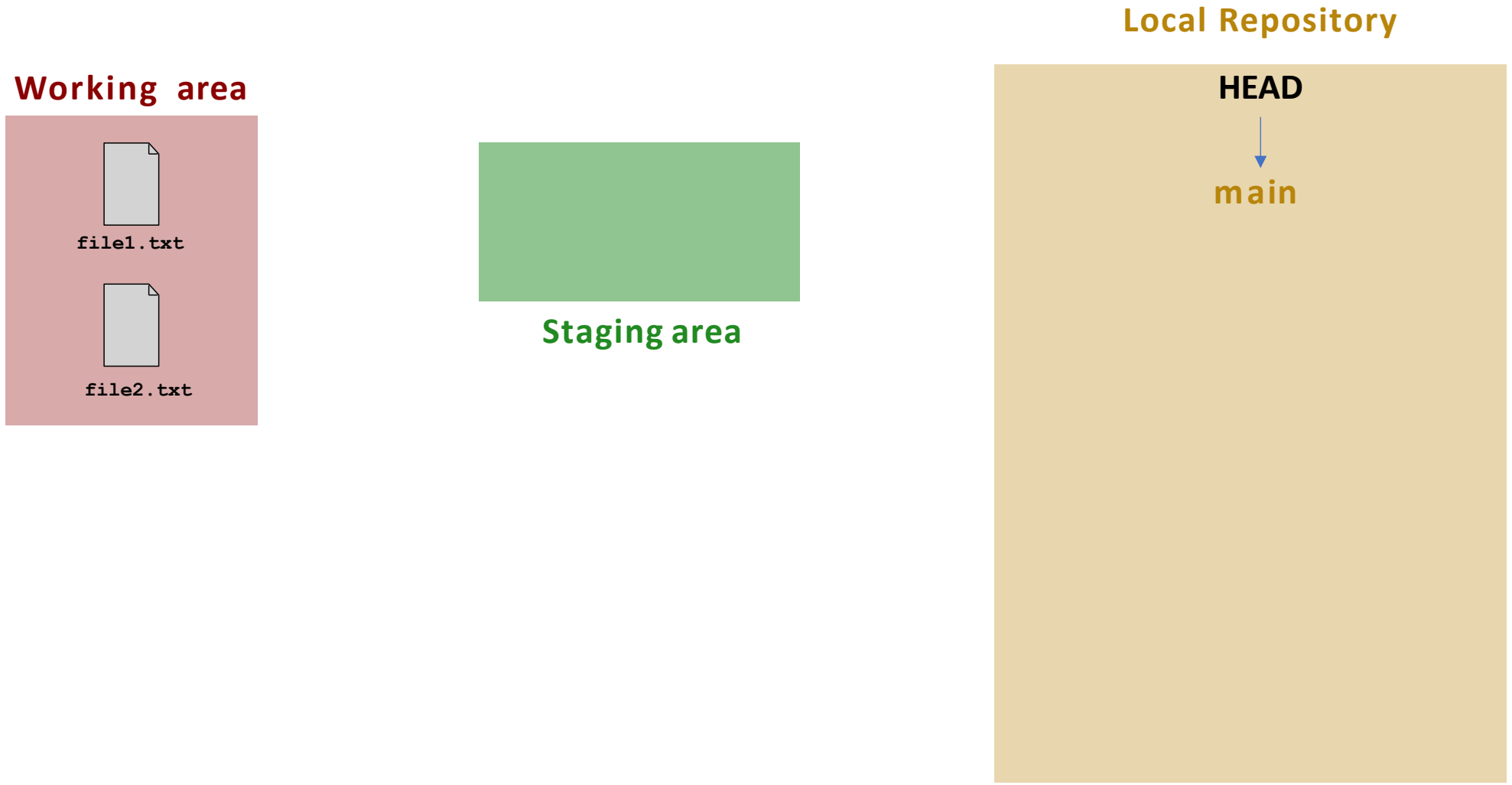
---

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

Git operations – locally

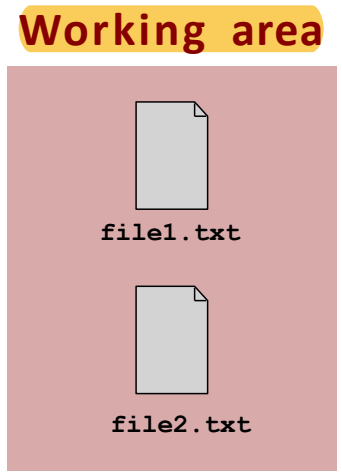


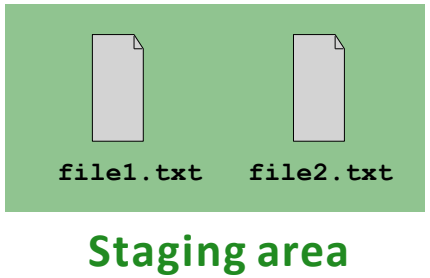
Git operations – locally

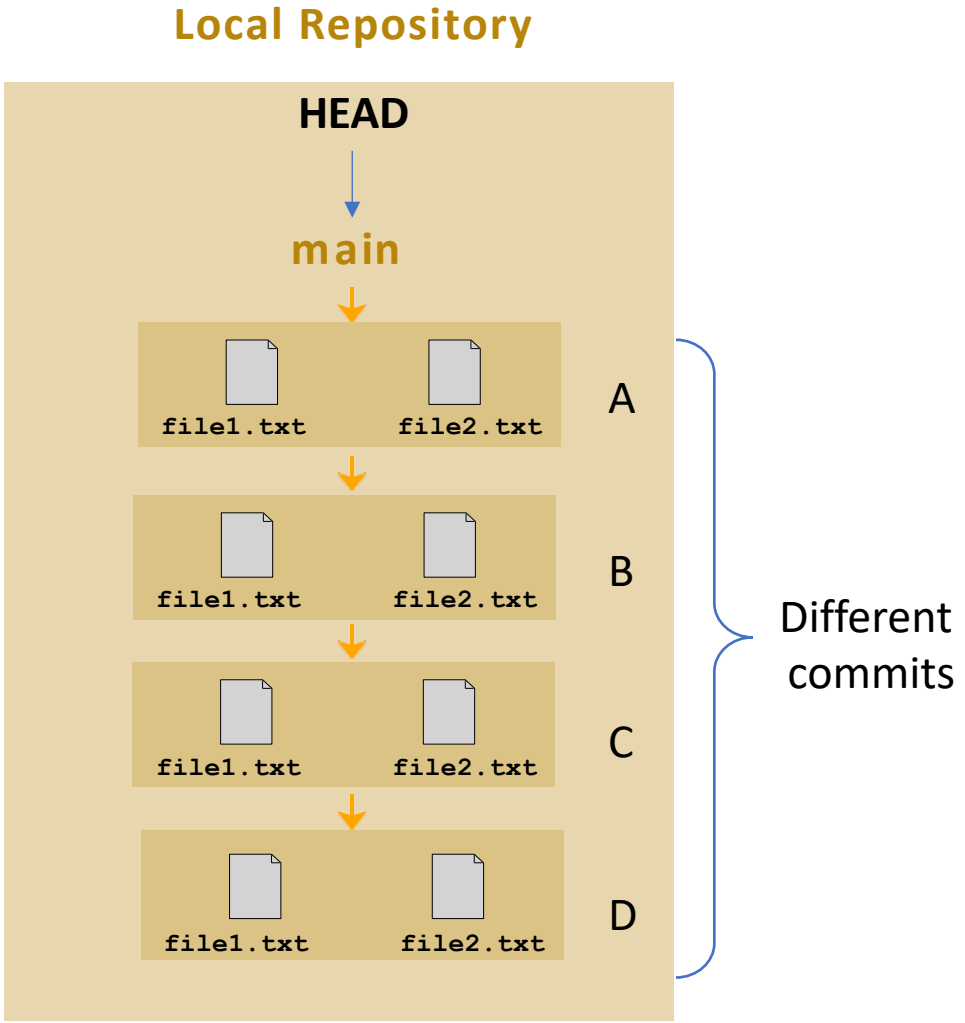


Git operations – locally

git init ->create a local repo



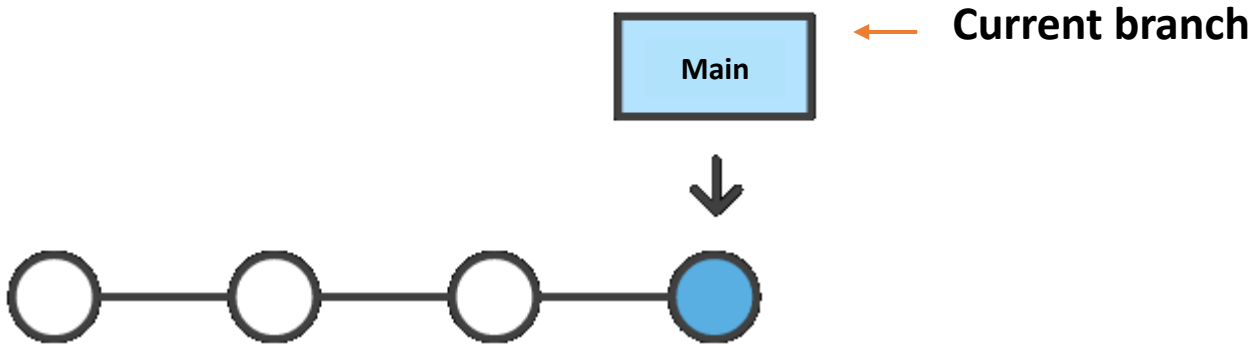




git clone <repo> <target_repo>

Git

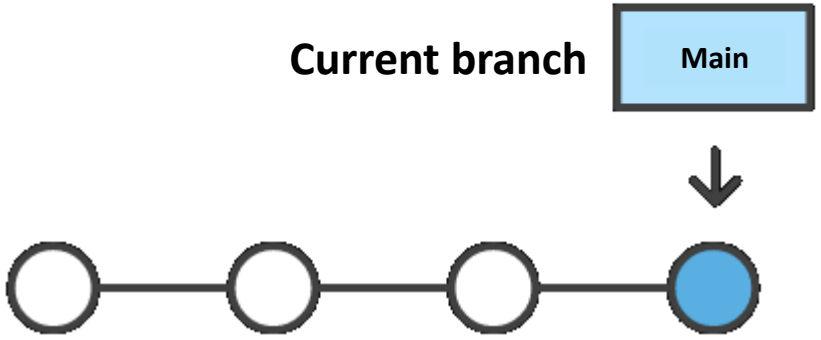
- Consider a supermarket program
- How to add some new features?
  - Check sold items
  - Check duplicates
  - Find total amount

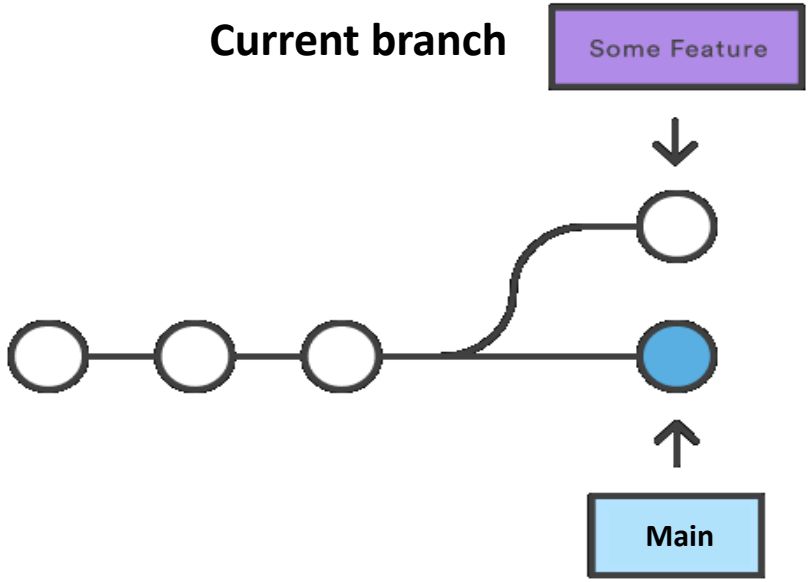


- Concern
  - Decided not to go with the new features at a later stage
  - Do NOT want to change the files for the working program

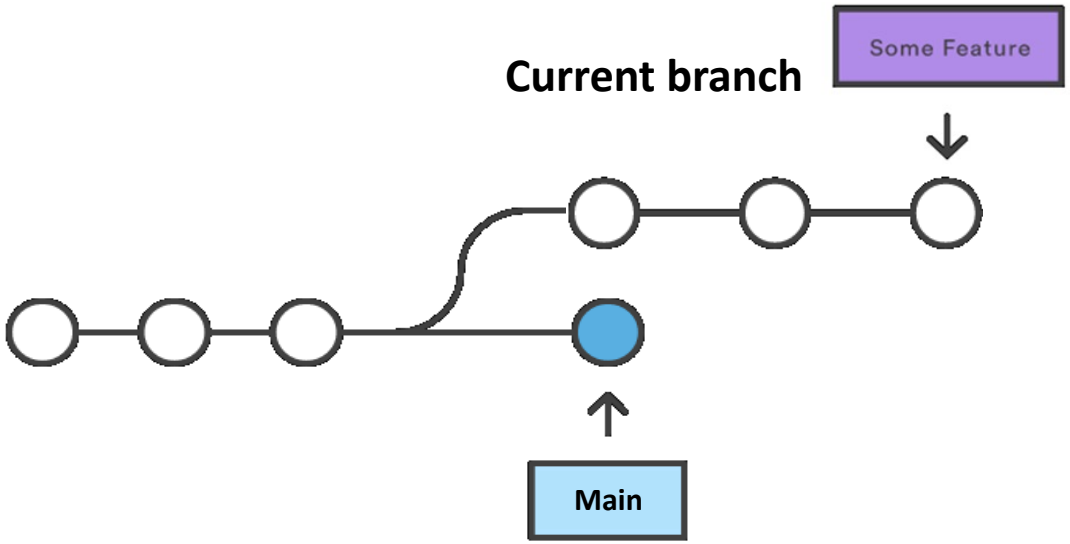
Git: branches

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

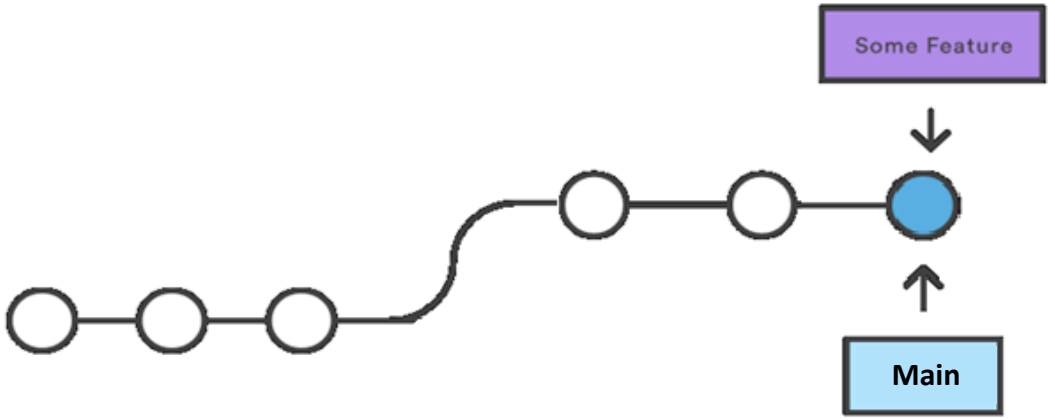




Git: branches



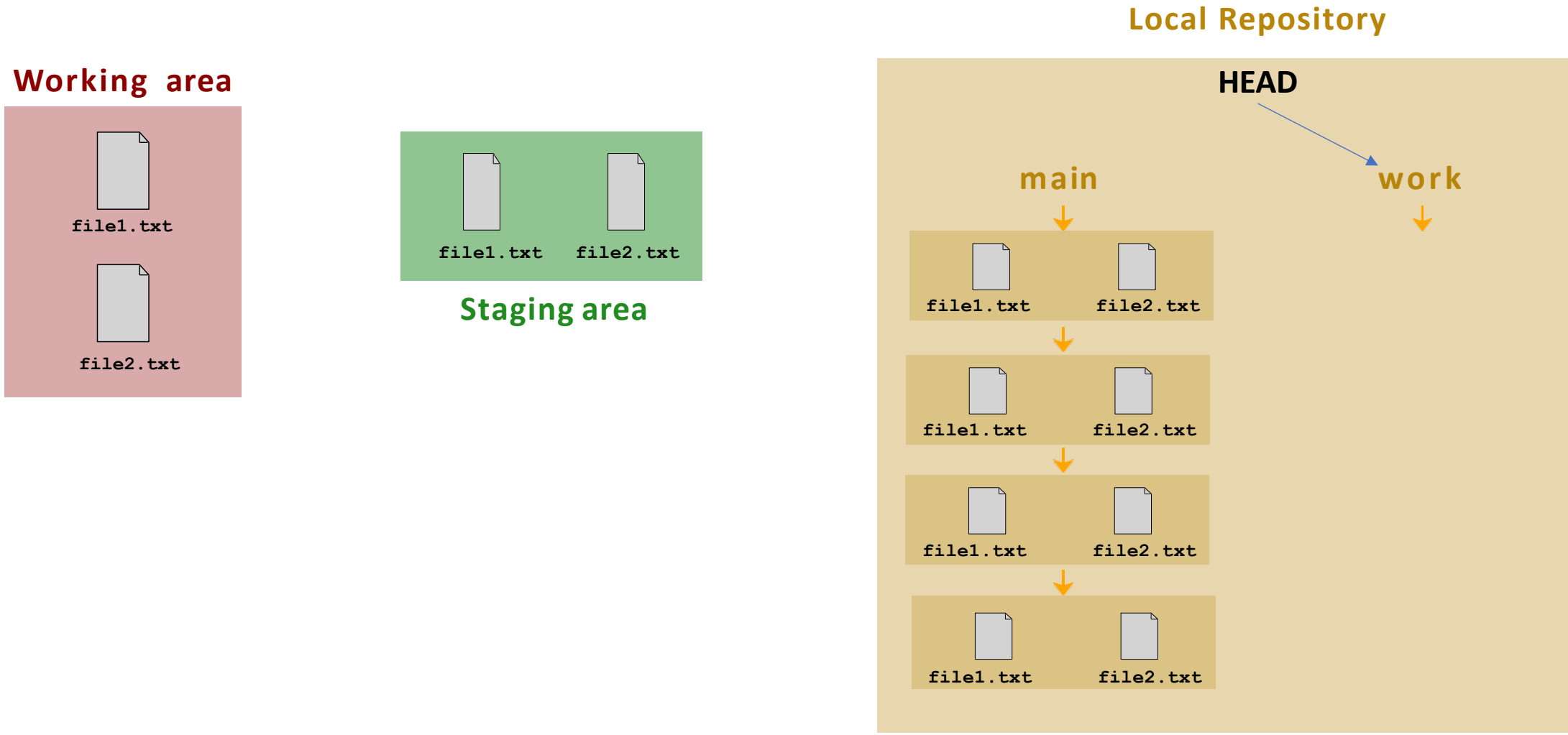
Add and commit changes to the branch



Once done, move back and merge to master:

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

Git operations – locally



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 fne 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
```

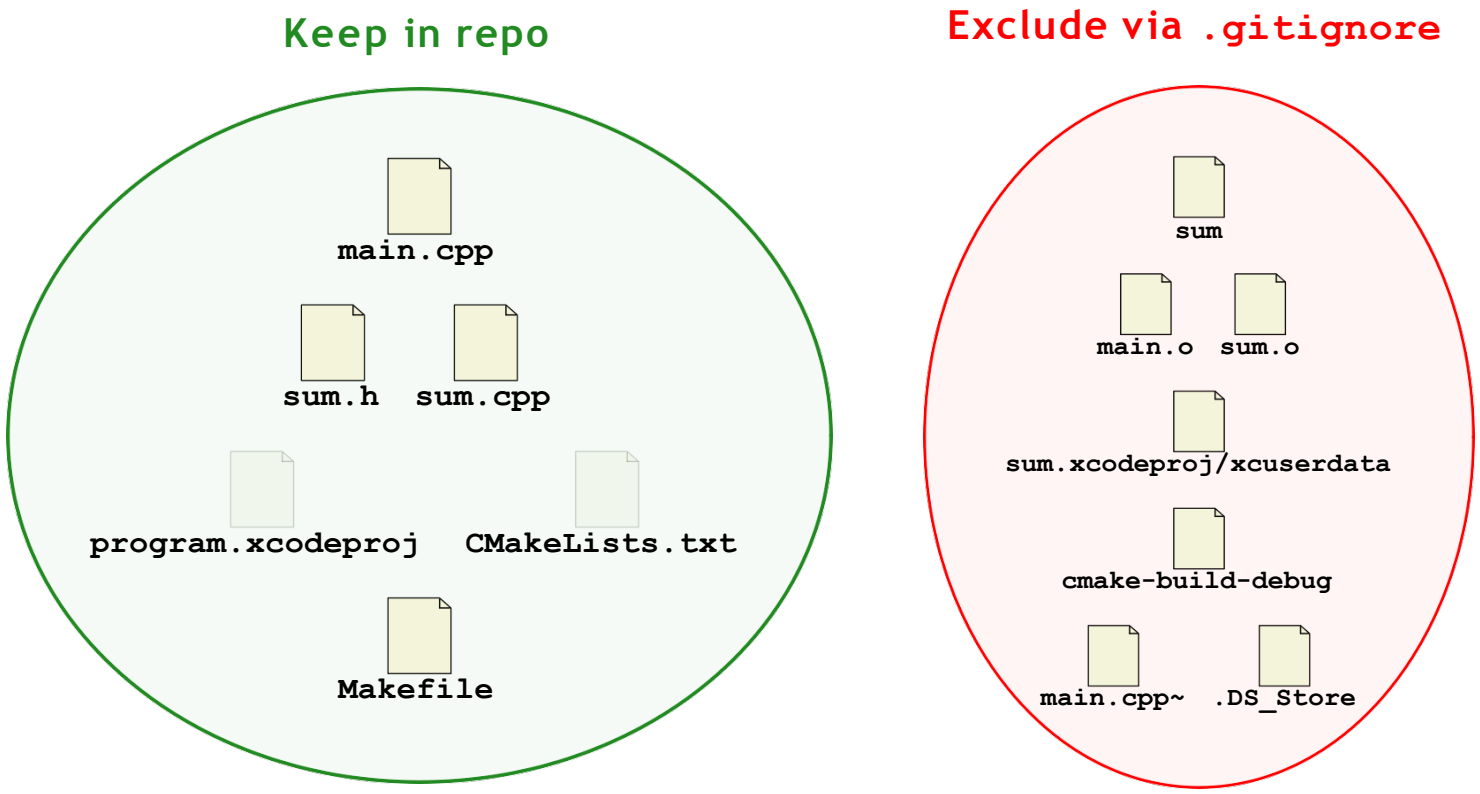
- Make changes and periodically save work

```
$ 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



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 fles there incuding **.gitignore**, and use

Assuming local repo exists

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

New name of the remote repo

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

Or: after creating the repo online, git clone …

-force -upstream

(link your local master to the remote master)

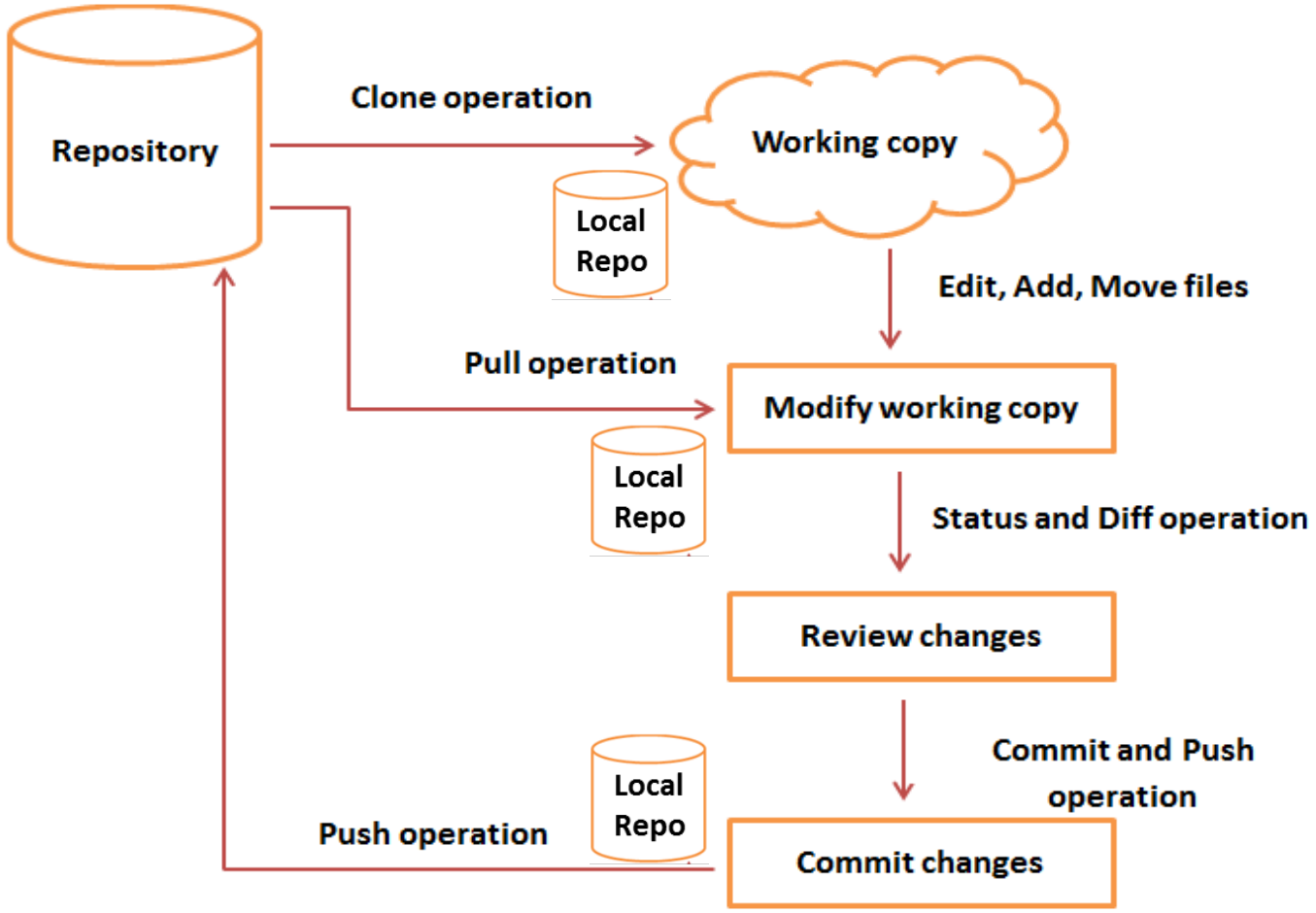
Git and github: Sketching full example

- 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 fles there incuding **.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:

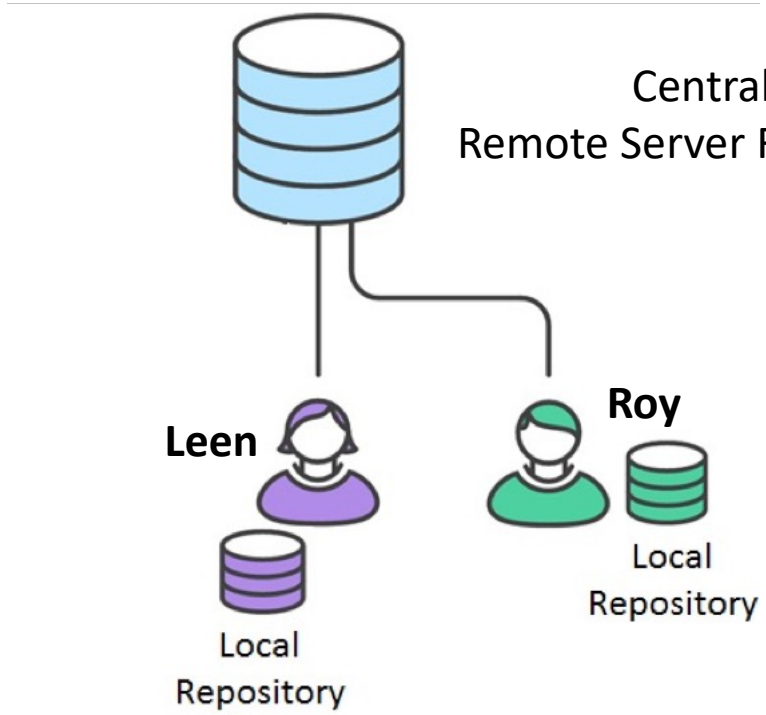
- ***git push***: apply changes in local repo to central repo

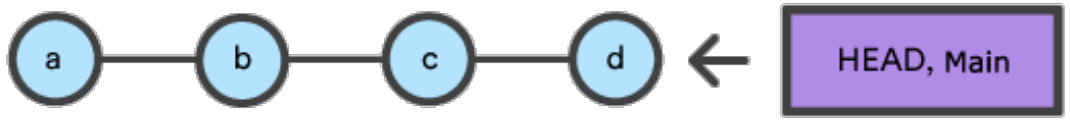
---

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

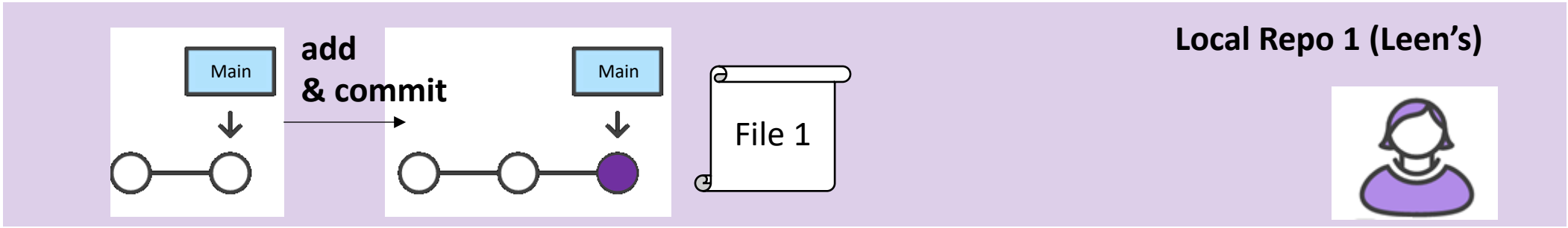
Git and github

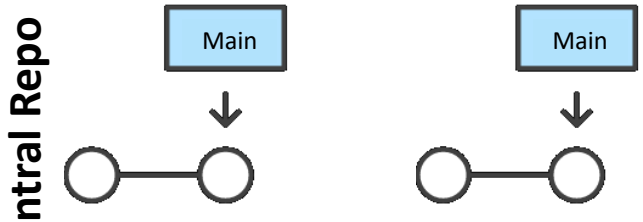
- What could go wrong?
- Scenario 1
- Scenario 2

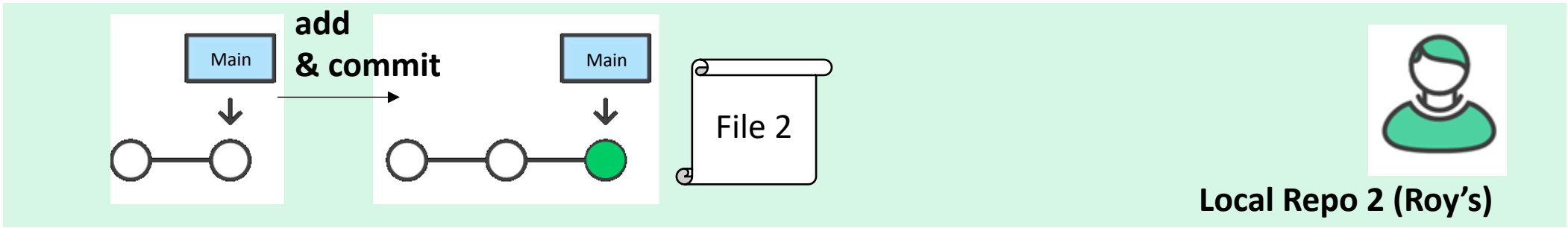




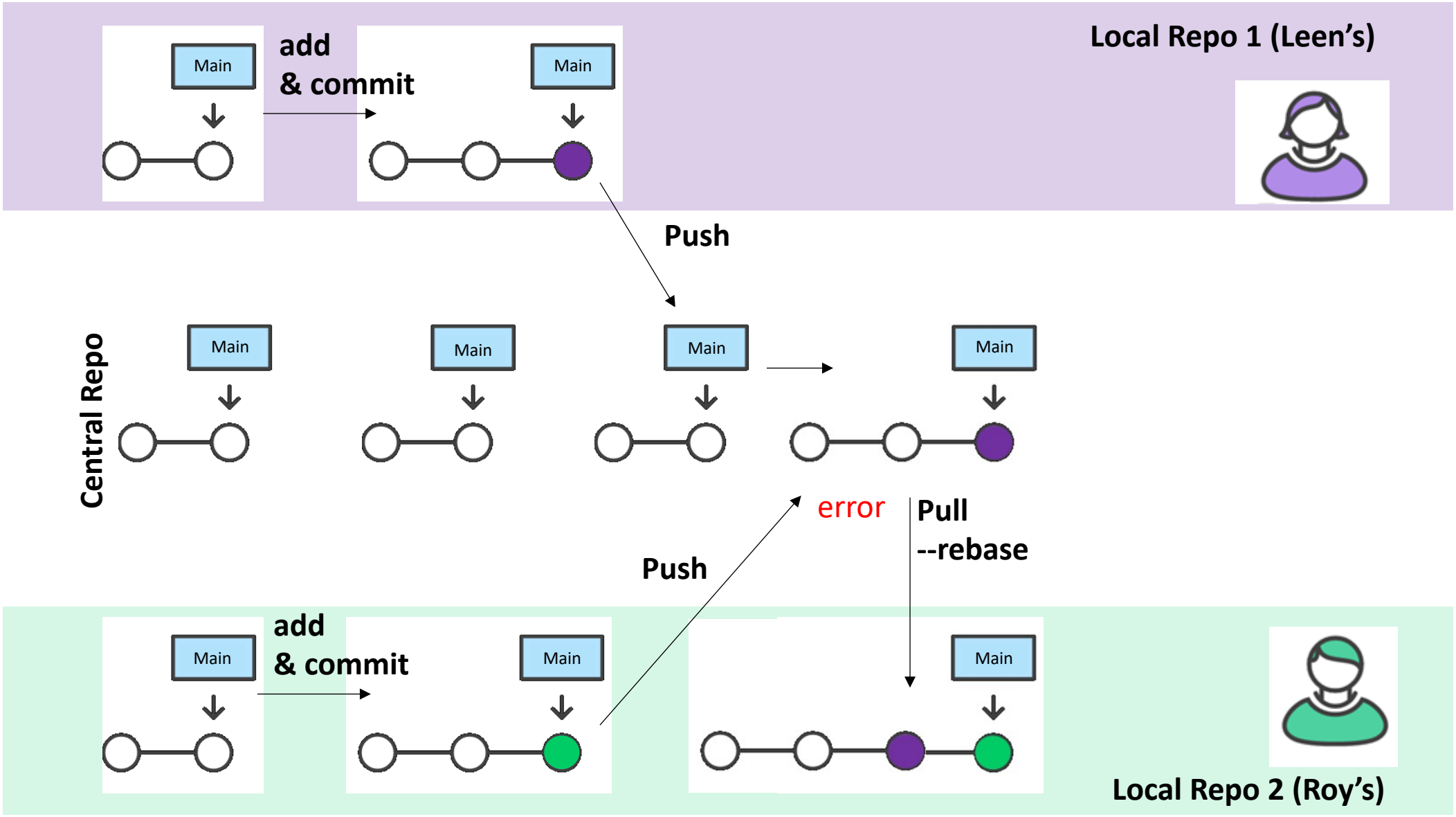
Scenario 1


Main
add
& commit
Main
File 1
Local Repo 1 (Leen's)

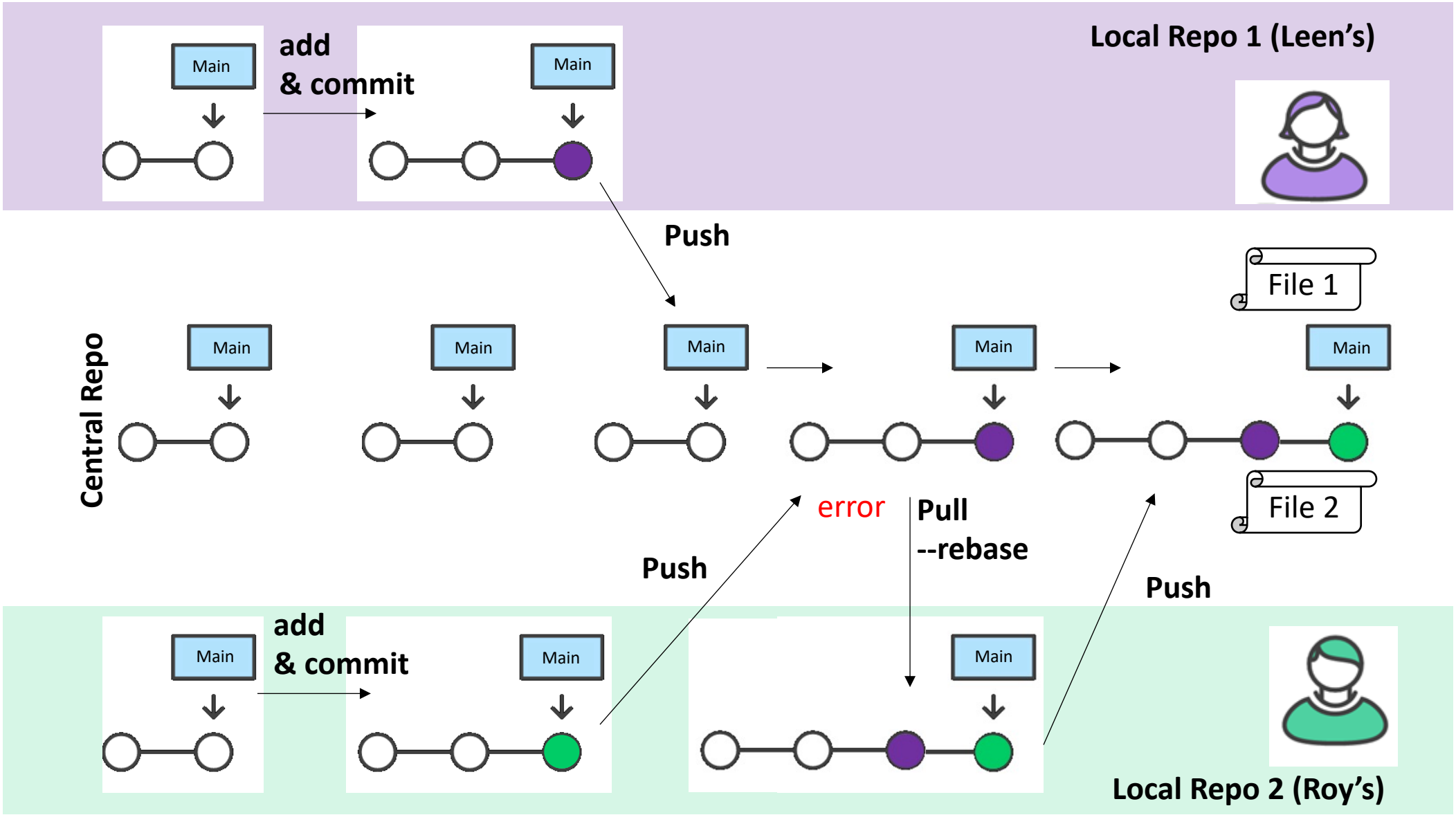

Central Repo
Main
Main


Main
add
& commit
Main
File 2
Local Repo 2 (Roy's)

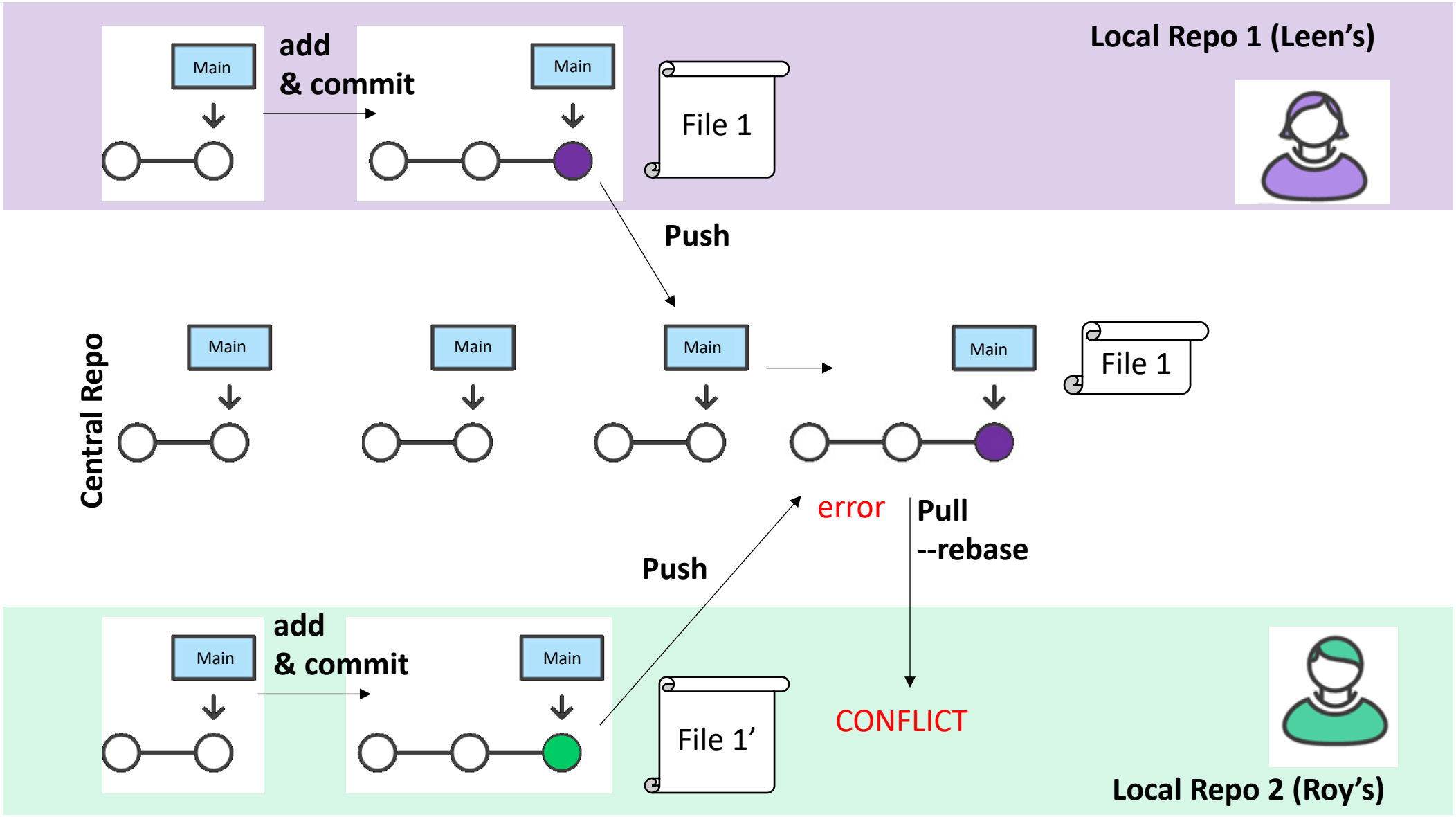
Scenario 1


Local Repo 1 (Leen's)
Main
add
& commit
Main
Push
Central Repo
Main
Main
Main
Main
error
Pull
--rebase
Push
Main
add
& commit
Main
Main
Local Repo 2 (Roy's)

Scenario 1


Local Repo 1 (Leen's)
Main
add
& commit
Main
Push
Central Repo
Main
Main
Main
Main
Main
File 1
File 2
error
Pull
--rebase
Push
Push
Main
add
& commit
Main
Main
Local Repo 2 (Roy's)

Scenario 2


Local Repo 1 (Leen's)
Main
add
& commit
Main
File 1
Push
Central Repo
Main
Main
Main
Main
File 1
error
Pull
--rebase
Push
Main
add
& commit
Main
File 1'
CONFLICT
Local Repo 2 (Roy's)

Scenario 2

Local Repo 1 (Leen's)

Central Repo

File 1

Main

error

Pull --rebase

File 1 Merged

CONFLICT

git status

File 1'

Edit conflicting files

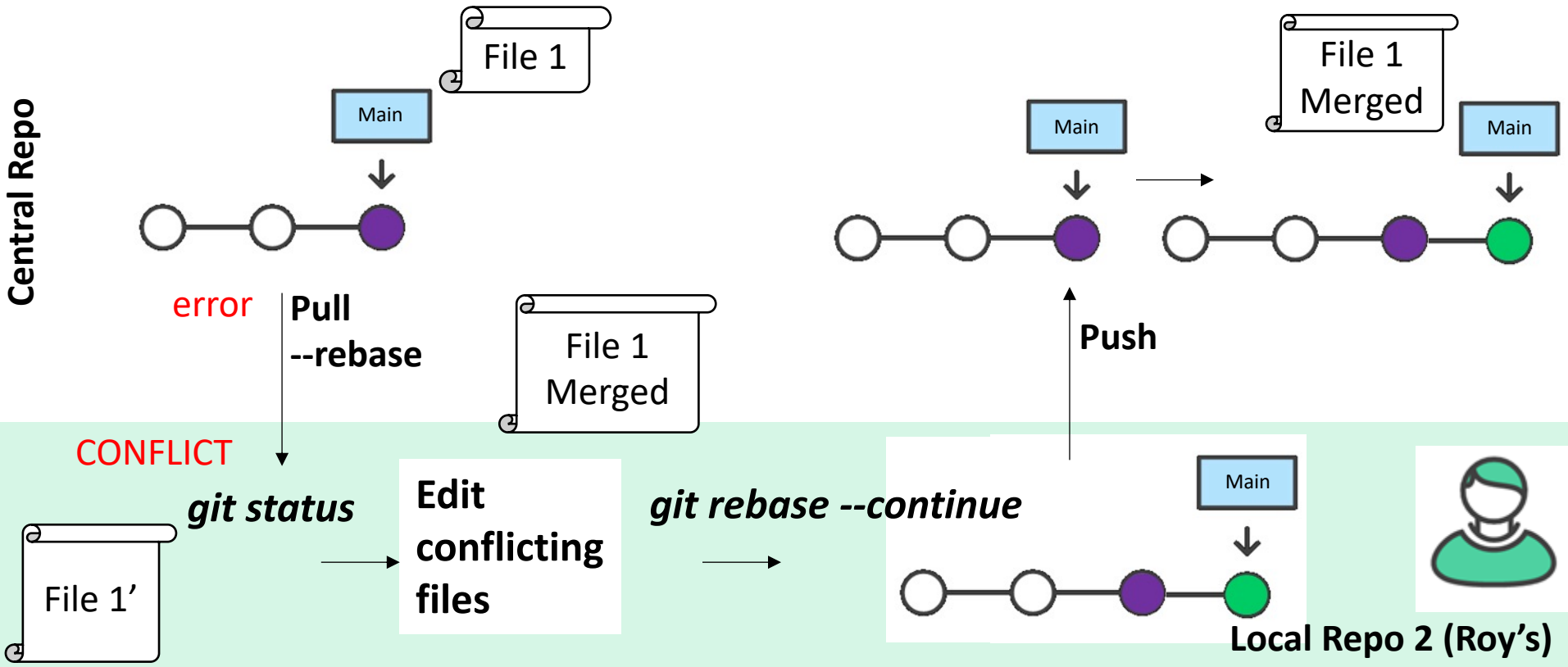
git rebase --continue

Main

Local Repo 2 (Roy's)

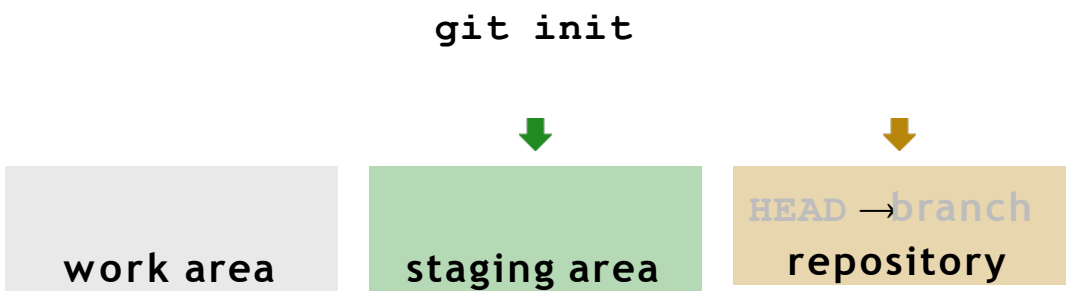
Scenario 2

Rebasing is the process of combining or moving a sequence of commits on top of a new base commit. Git rebase is the linear process of merging.

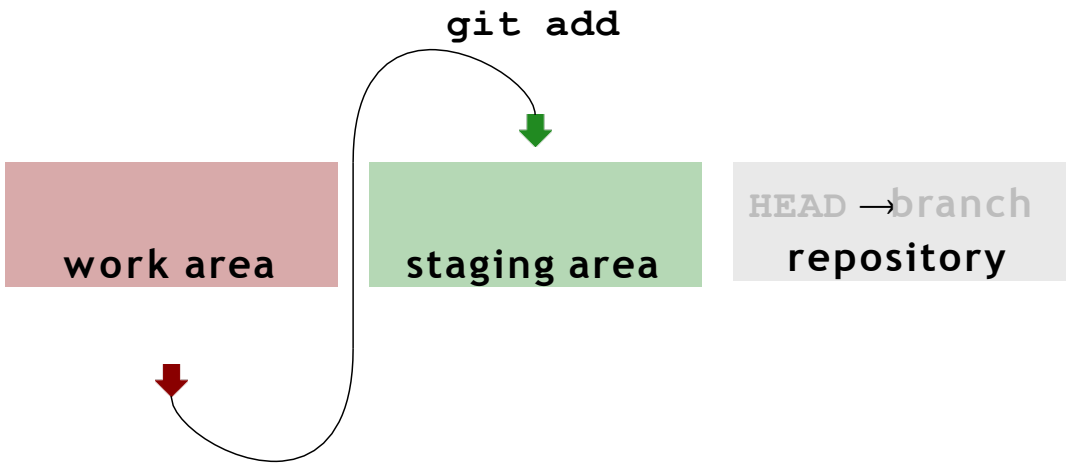


Backup slides: More Git

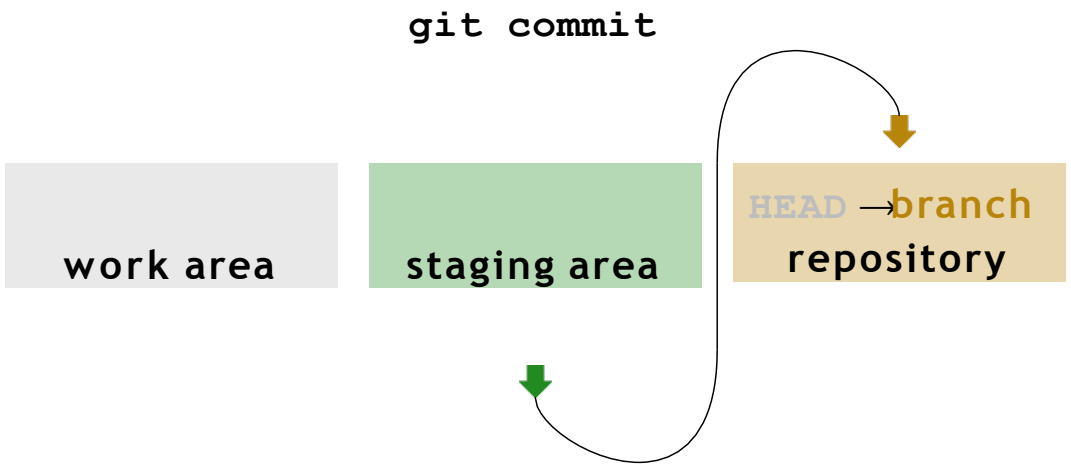
Git Command



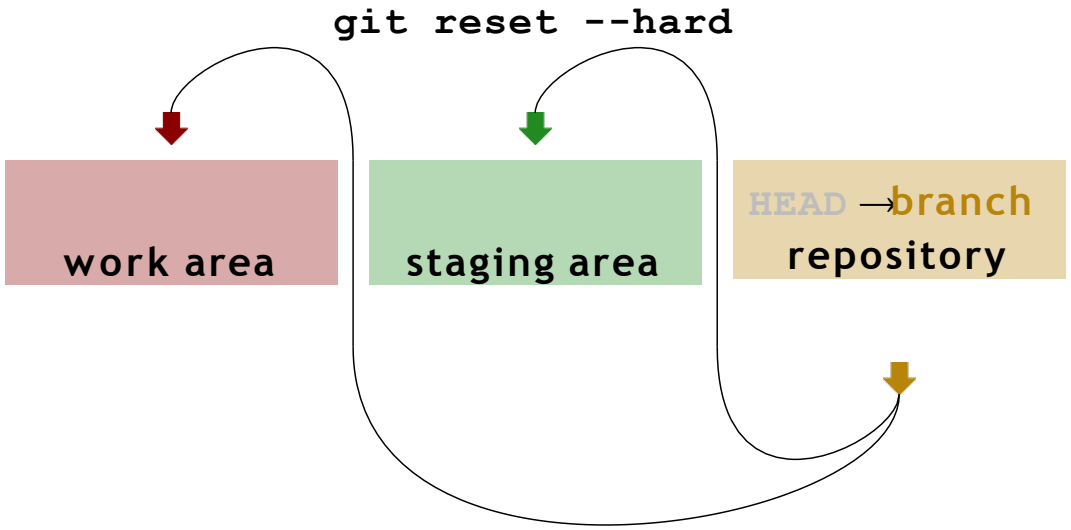
Git Command



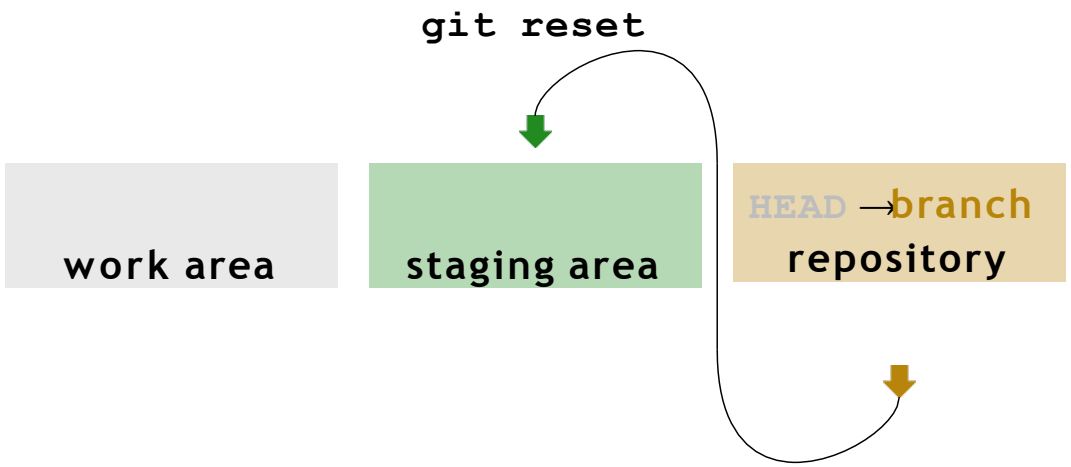
Git Command



Git Command


git reset --hard
work area
staging area
HEAD → branch
repository

Git Command



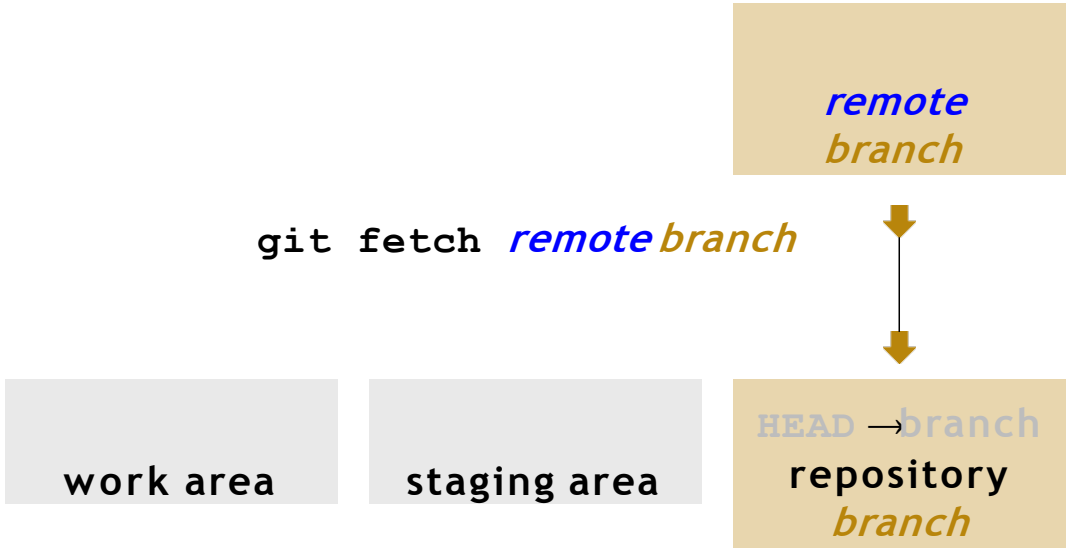
Git Command


origin
git fetch
work area
staging area
HEAD →branch
repository

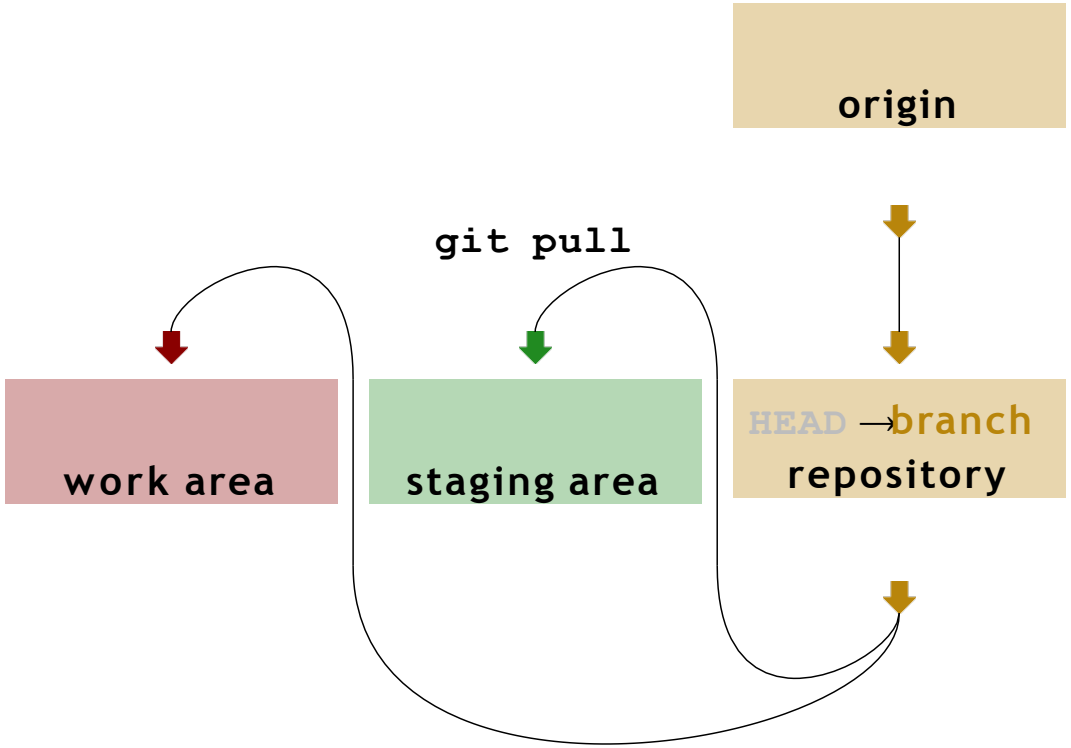
Git Command


remote
git fetch remote
work area
staging area
HEAD → branch
repository

Git Command

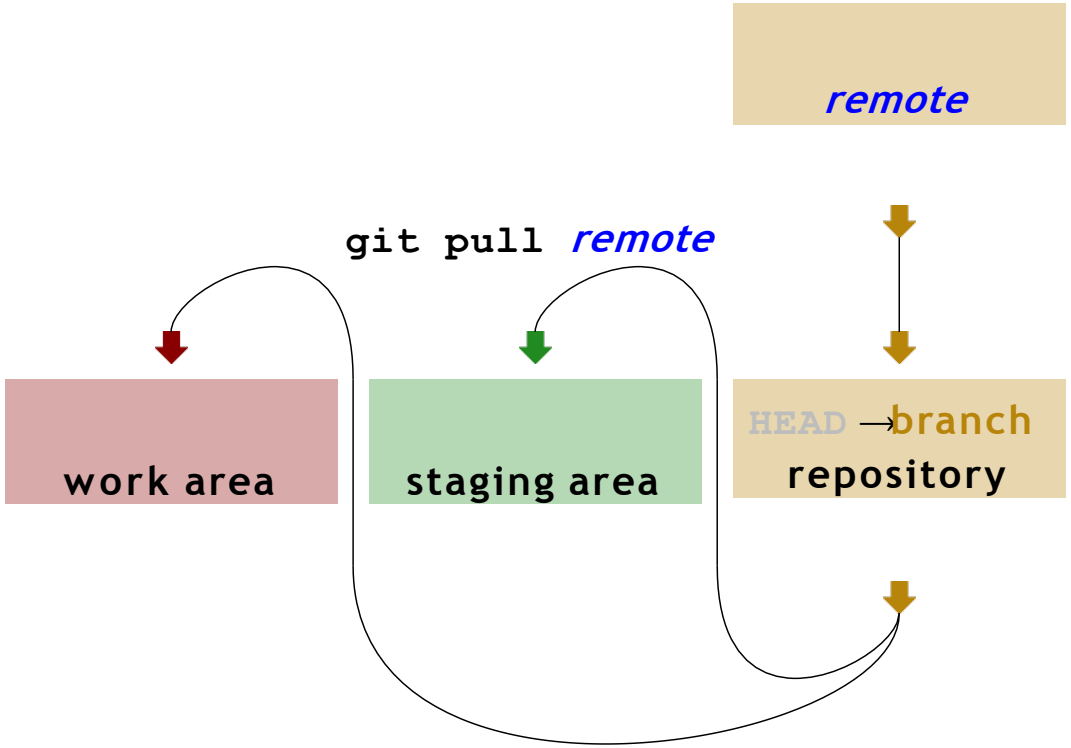

remote
branch
git fetch remote branch
work area
staging area
HEAD → branch
repository
branch

Git Command



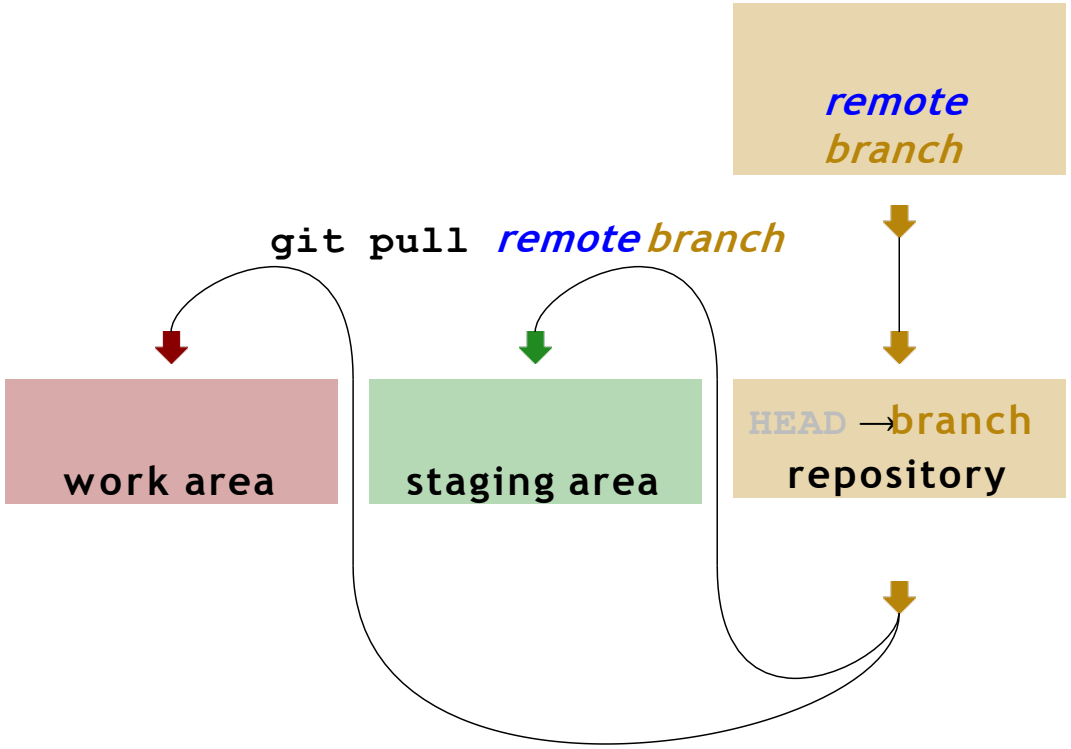
git pull is essentially **git fetch && git merge**

Git Command



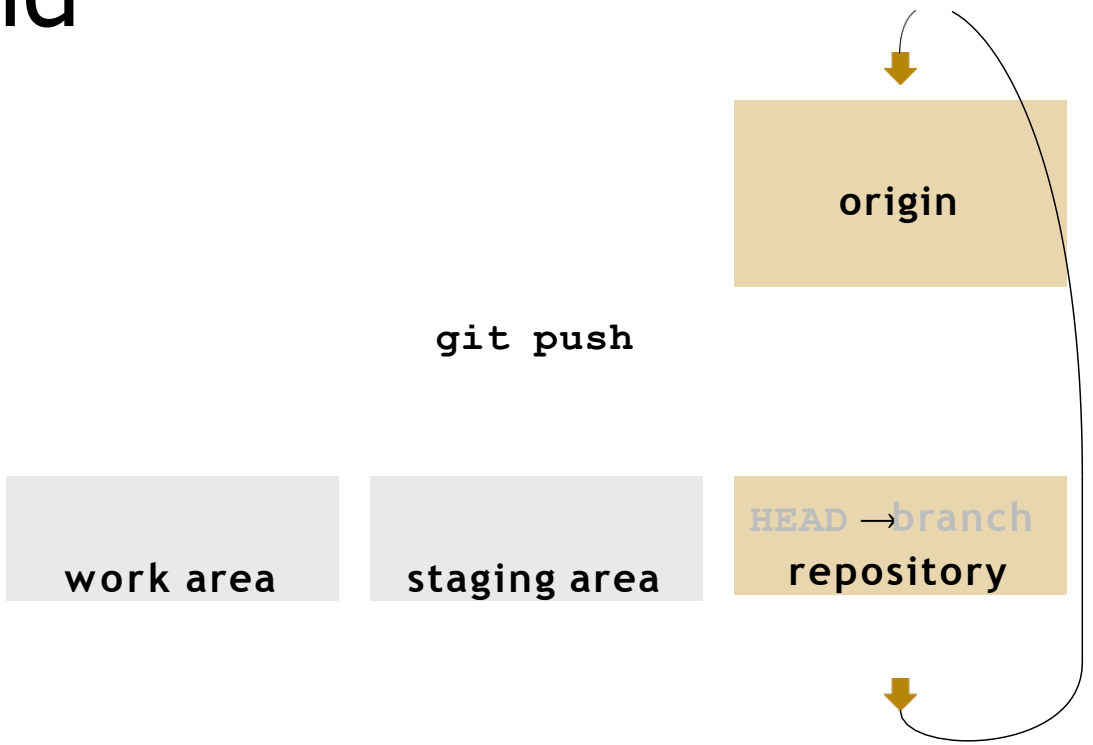
git pull is essentially **git fetch && git merge**

Git Command

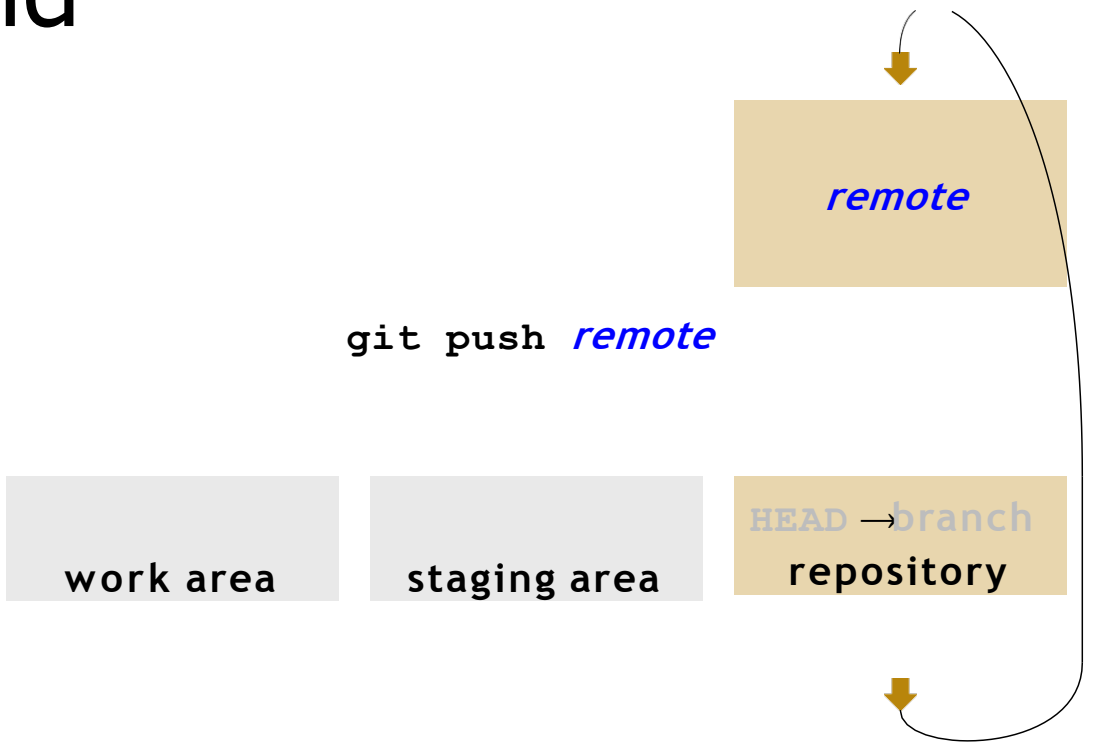


git pull is essentially **git fetch && git merge**

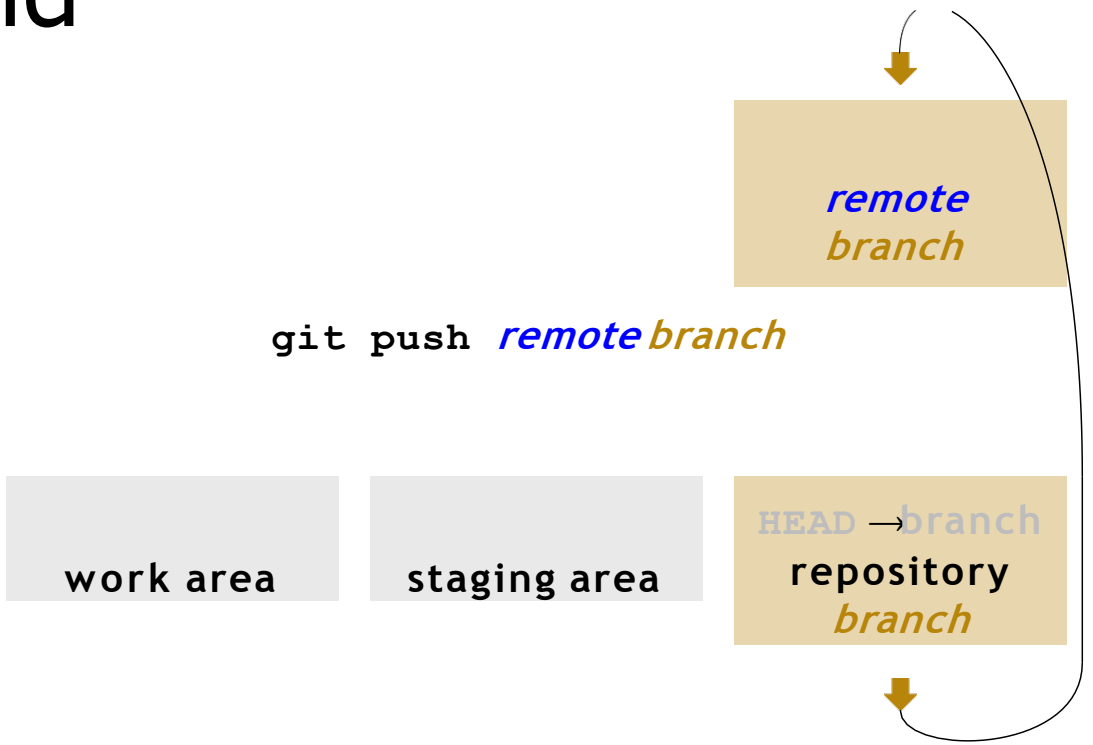
Git Command


origin
git push
work area
staging area
HEAD → branch
repository

Git Command


remote
git push remote
work area
staging area
HEAD → branch
repository

Git Command


remote
branch
git push remote branch
work area
staging area
HEAD → branch
repository
branch