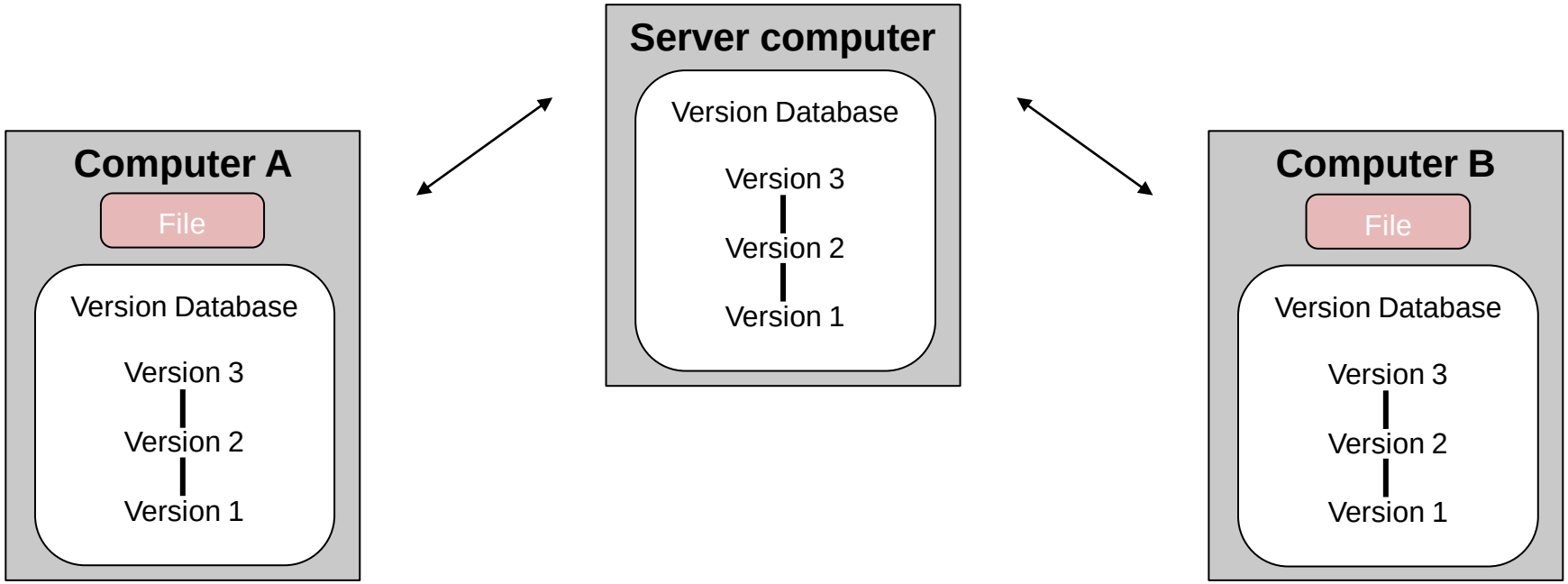
Git
(Version-Control System)

Practical Class 3

Systems and Storage Laboratory
Department of Computer Science and Engineering
Chung-Ang University

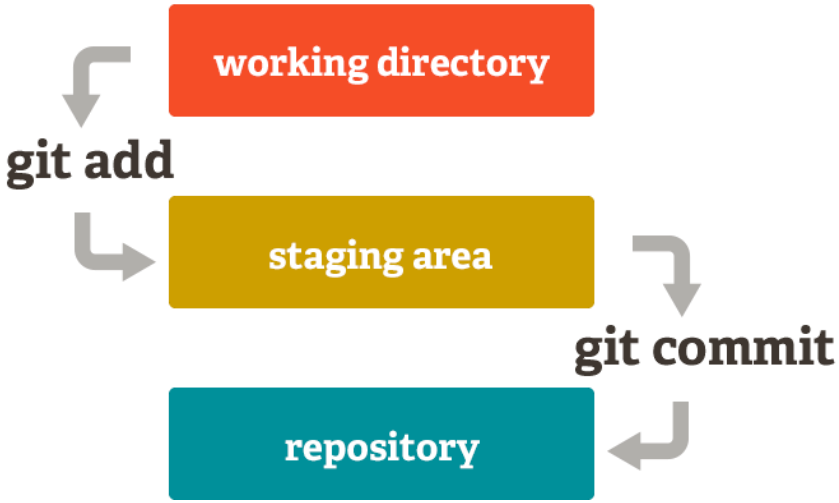
What is git?

- **Git is Distributed Version Control System(DVCS)**
  - No need to connect to the central server
  - Every copy of the repository can serve either as a server or as a client
  - Git tracks changes, not versions



Local Git areas

- **In your local copy on git, files can be:**
  - In your local repo
    - **Committed**
  - Checked out and modified, but not yet committed
    - **Working copy**
  - Or, in-between, in a **staging** area
    - **Staged files** are ready to be committed
    - A commit saves a snapshot of all staged state

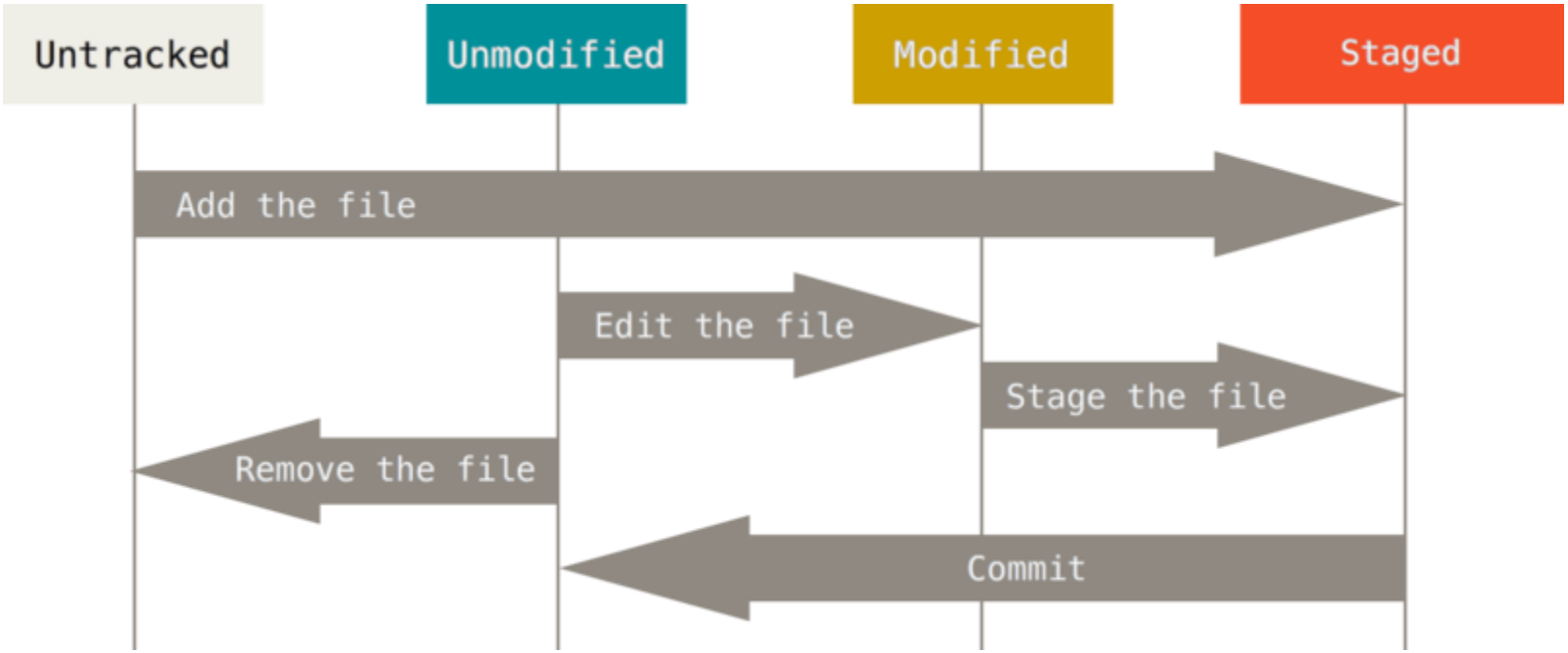


[Fig 1] Staging Area

[1] https://git-scm.com/about/staging-area

Basic Git workflow

1. **Modify** files in your working directory
2. **Stage** files, adding snapshots of them to your staging area
3. **Commit**, which takes the files in the staging area and stores that snapshot permanently in your Git directory



[Fig 2] The lifecycle of the status of your files

[2] https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository

Git commit checksums

- In Git, each user has their own copy of the repo and commits changes to their local copy of the repo before pushing it to the central server.
- Git generates a unique **SHA-1 hash** for every commit
  - Refers to commits by this ID rather than a version number
- We use this hash value to switch between different versions
  - We can use the first 7 characters of the hash value

Install and Setup

- **Install**

```
$ sudo apt-get install git
```

- **Setup your user name and user email**
  - To apply for the whole system

```
$ git config --global user.name "<User Name>"
$ git config --global user.email "<User Email>"
```

  - To apply just for this repository

```
$ git config --local user.name "<User Name>"
$ git config --local user.email "<User Email>"
```

- **To review git configurations**

```
$ git config --list
```

Start git – Create a local repository

1. **Move to home directory**

```
$ cd ~
```

2. **Make a directory for the repository**

```
$ mkdir ~/<repository name>
```

3. **Move to repository**

```
$ cd <repository name>
```

4. **Make a directory and a file**

```
$ mkdir <directory name>
$ touch <file name>
```

Start git – Create a local repository

5. **Initialize local git repository**

```
$ git init
```

6. **Add every file to repository**

```
$ git add .
```

7. **Commit to local git repository (Make first revision)**

```
$ git commit –m “<description for this commit>”
```

8. **See log file**

```
$ git log
```

- **Every version info is saved in a `.git` directory.**
  - `.git` directory exists in the repository directory
  - It maintains the settings and history of the repository

Start git – Download a remote repository

- To **clone a remote repo** into your current directory:

```
$ git clone <url> [<local directory name>]
```

- We call the repository that we want to clone a **remote repo**
  - Repository in your local machine can also become a remote repository

- Check **remote repository information**:

```
$ git remote
```

Start git – Review and commit modifications

- **When users change files, the difference occurs**

- **Check the changes and commit them to the repository**
  1. First, make some changes to the files
  2. Show the difference between what are staged and what are modified but unstaged yet

```
$ git diff
```

  3. Commit modifications

```
$ git commit –m “<description of commit>”
$ git commit -a –m “<description of commit>”
```

  - ✓ **-a** : only commit revision files

Start git – Current status and history

- Check current status of repository

```
$ git status
```

- Check history logs of commits

```
$ git log
```

Revert Changes

Clean

- Remove untracked files from the working tree

```
$ git clean –n
```

✓ **-n** : Don’t actually remove anything, just show what would be done

```
$ git clean –f
```

✓ **-f** : force delete

Revert Changes

- **If modifications have already been staged by 'add'**

```
$ git reset [<mode>] [<commit>]
```

  - e.g. To unstage a file

```
$ git reset HEAD -- <filename>
```

- **Undo the changes by "checkout"**
  - e.g. Undoes your changes using the staged files

```
$ git checkout -- <filename>
```

Tagging in Git

- **Git provides option that can set tags to certain versions existing in history.**
  - It makes user easily refer former sources.
  - Generally, tags are used freely released versions.

- **Make tag**

```
$ git tag <tag name>
```

- **Push tag to remote repository**

```
$ git push <remote repository name> <tag name>
```

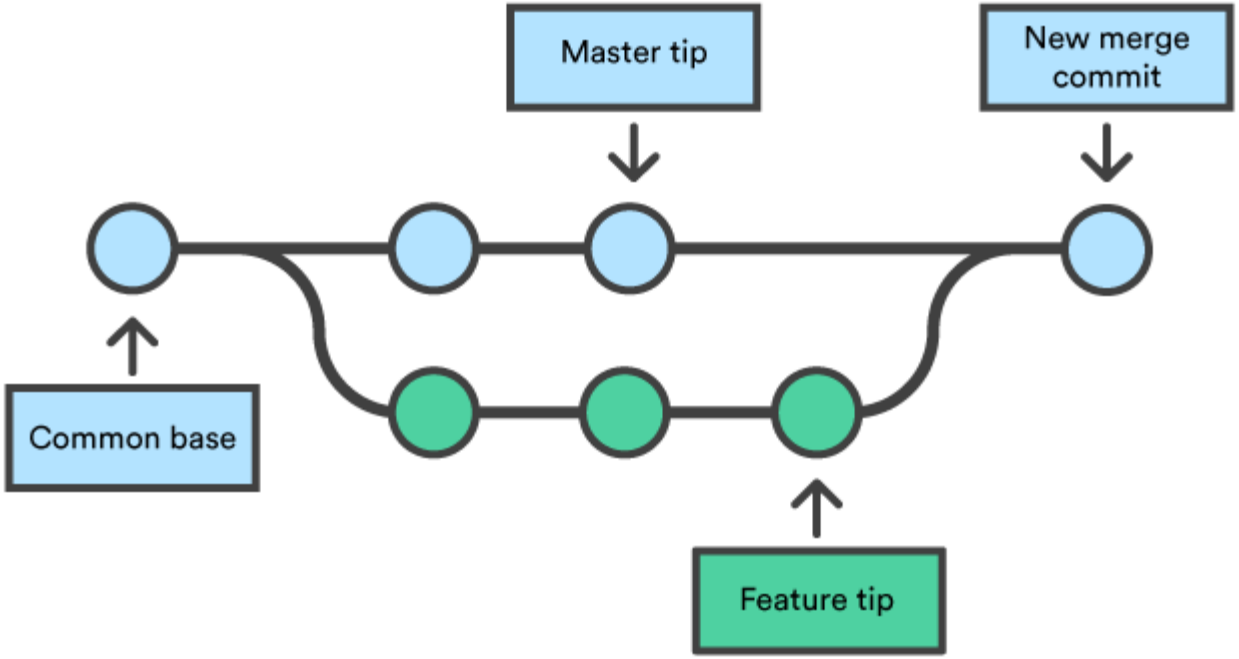
- **Delete tag**

```
$ git tag -d <tag name>
```

Branch

Branch

- Git provides a feature for making branches
- You can treat branches as independent copies
  - Users can modify each branch independently



Branch

❖ Show list of branches

```
$ git branch
```

```
$ git branch –a
```

✓ **-a** : List all the branches including remote branches

❖ Create branch

```
$ git branch <branch name>
```

❖ Switch to other branches

```
$ git checkout <branch name>
```

❖ Delete branch

```
$ git branch –d <branch name>
```

✓ **-d** : Delete branch <branch name>

Merge

❖ Merge

- Join development histories of the subject branch to current branch

```
$ git merge <subject branch>
```

  - Assume the following history exists and the current branch is "**main**":

```
          A---B---C    topic
         /
D---E---F---G        main
```

  - Then "**`git merge topic`**" will replay the changes made on the **topic** branch since it diverged from **master** (i.e., **E**) until its current commit (**C**) on top of master, and record the result in a new commit along with the names of the two parent commits and a log message from the user describing the changes.

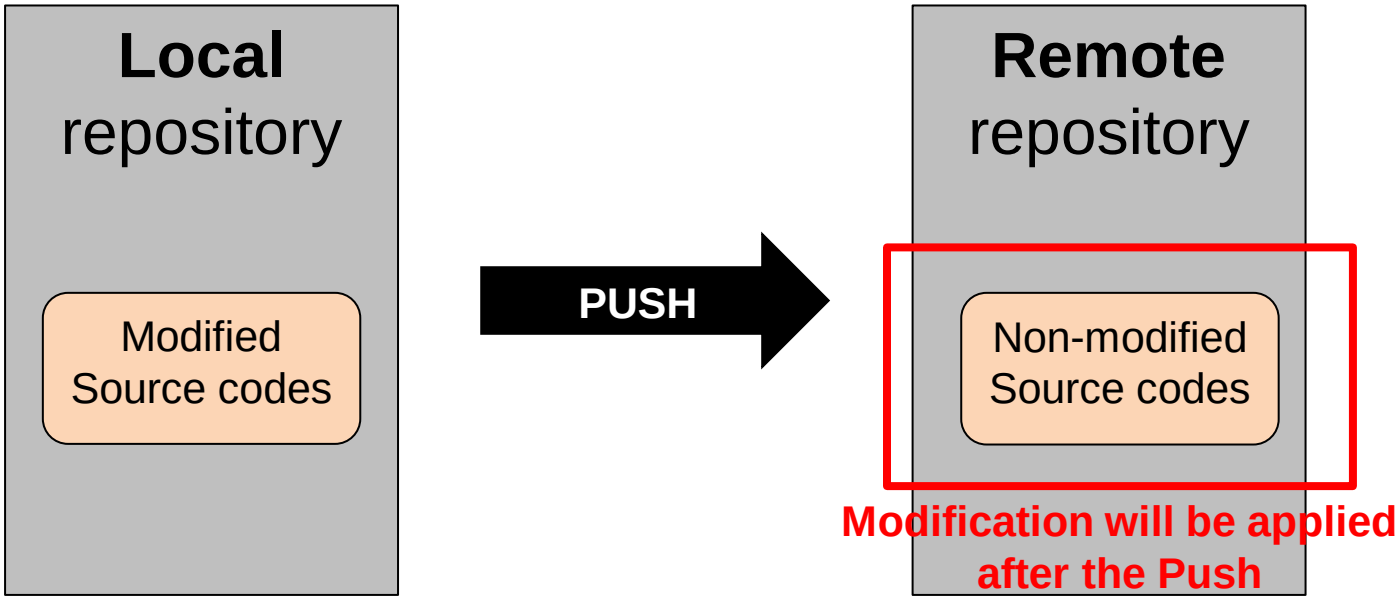
```
          A---B---C    topic
         /         \
D---E---F---G---H    main
```

Working with remote repo: Push and Pull

Push

- Upload modified files to the remote repository

```
$ git push <name of remote repository> [<branch name>]
```

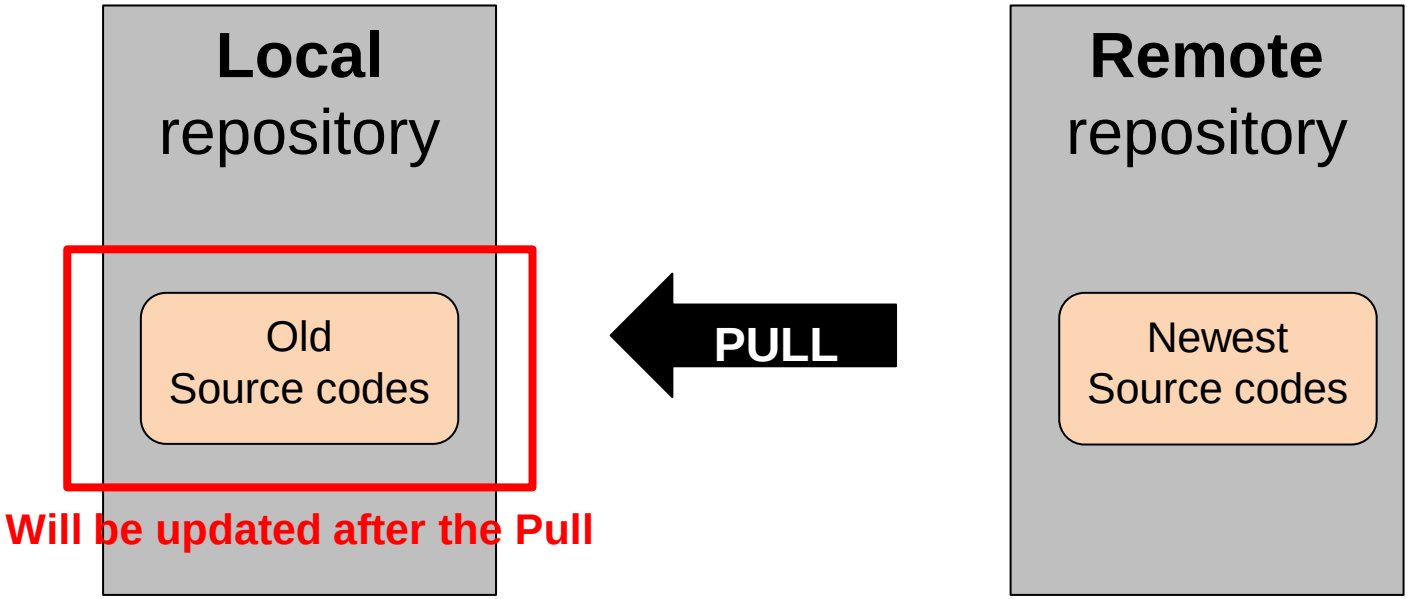


Working with remote repo: Push and Pull

Pull

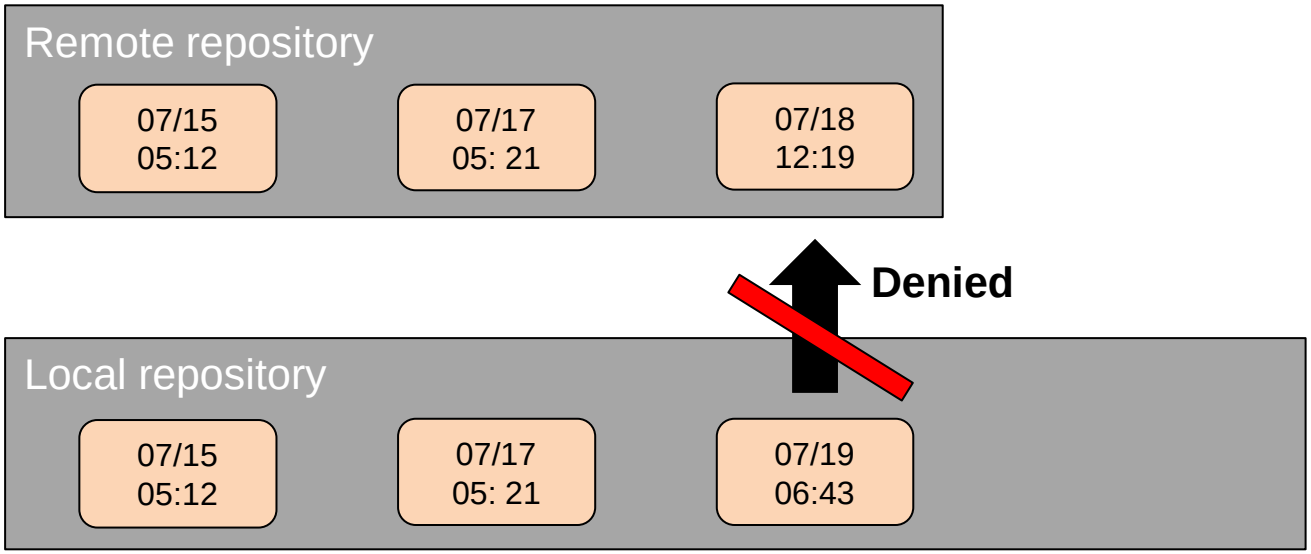
- Update with remote repository to local repository

```
$ git pull <name of remote repository> [<branch name>]
```



Merging from remote repo

- When pushing the local repo to the remote side, if the local side history is not synced with the remote history, the push operation is denied.
- In this case, **`merge`** helps us to update the local repo with other users' update history on the remote side.
- Otherwise the history of the commit from other users will be disappeared, so we must **`merge`** before overwriting it



Merging from remote repo

- **We use pull to sync the history between remote and local repo**
  - **`git pull`** is the combination of
    `git fetch` and **`git merge`**