Getting started with GitHub & GitHub Desktop

TUFTS
DATALAB

SHIRLEY LI, PHD
BIOINFORMATION
TUFTS TECHNOLOGY SERVICES
(SOME ORIGINAL CONTENT BY UKU-KASPAR UUSTALU AND KYLE MONAHAN)

Content developed by

Xue (Shirley) Li

Bioinformatician

xue.li37@tufts.edu

Kyle Monahan

Manager of Data Science Services

Kyle.Monahan@tufts.edu

Uku-Kaspar Uustalu

Senior Data Science Specialist

Uku-Kaspar.Uustalu@tufts.edu

Delilah Maloney

Sr. High Performance Computing Specialist

Delilah.Maloney@tufts.edu

Yucheng Zhang

Bioinformatics Engineer

Yucheng.Zhang@tufts.edu

https://it.tufts.edu/research-technology-team

Overview

1. Data storage vs. GitHub Repositories
2. Git & Github & Version Control Basics
3. Navigating GitHub Desktop
4. Resolving Merge Conflicts
5. Understanding Branch Workflow
6. Best Practices for GitHub Usage

1. Data Storage vs. GitHub Repo

Local folder

- A local folder resides on your computer's hard drive or an attached storage device.
- No version control
- Cannot collaborate

Cloud folder (e.g., Dropbox and Box)

- Stored on servers accessible over the internet, allowing files to be accessed from multiple devices and locations.
- Basic version history is available (snapshot)
- Collaboration
- **Synchronization:** Files stored in cloud folders can be synchronized across all devices linked to the same cloud account. Changes made in one device will automatically update across all devices, ensuring consistency.

A local copy of a cloud-based storage

- The files stored in a cloud-based storage service (like Box, Dropbox, Google Drive, OneDrive, etc.) are also downloaded and stored on your own computer's hard drive or another storage device.

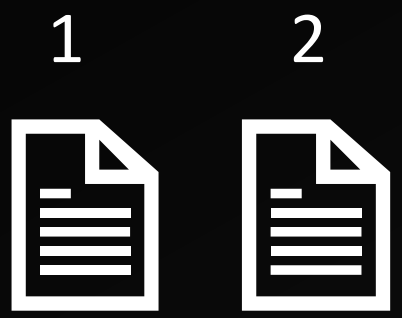
How changes are tracked

- In a shared folder, who ever made the last changes will be saved to cloud, and synced to all local copies.

Conflicts in Cloud Folder

- Conflicts in a cloud folder can occur when multiple users edit the same file at the same time and the system is unable to reconcile the changes automatically.
- Both versions will be saved. We need manually review conflict files.
- Best Practices: Coordinate with team members, agree on who will edit what and when.

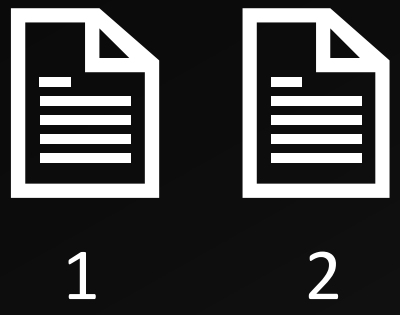
Cloud storage



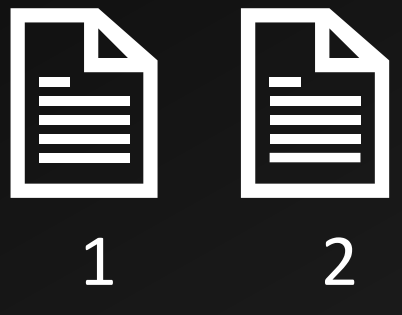
No Conflict

Local copy

You



Collaborator



No Conflict

Cloud storage

You

Collaborator

Local copy

No Conflict

Cloud storage

You

Collaborator

Local copy

No Conflict

Cloud storage

You

Collaborator

Local copy

With Conflict

Cloud storage

Local copy

You

Collaborator

With Conflict

Cloud storage

Local copy

You

Collaborator

With Conflict

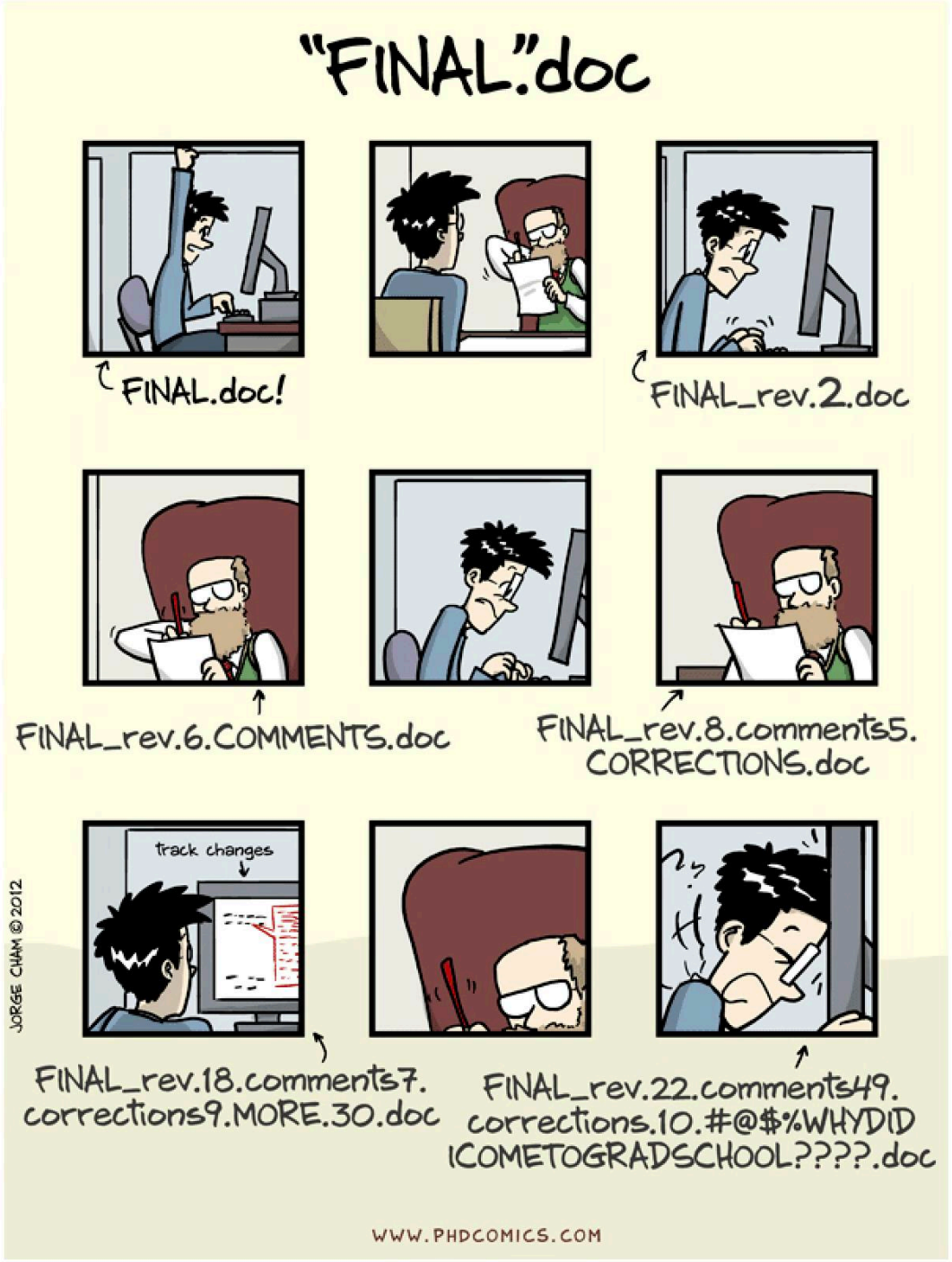
Cloud storage

Local copy

You

Collaborator

2. Git & GitHub & Version Control Basics


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.
CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.
corrections9.MORE.30.doc
FINAL_rev.22.comments49.
corrections.10.#@$%WHYDID
ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

Version Control

- Version control is a **record of who make changes to what, and when they did it**
- We can always undo
- Easier for **collaboration** without overwriting
- A key skill in **code&data management**!

What is Git?

- A version control system.
- Manage source code changes.
- Two key features: Commit and Branches.
- With Git, you can easily roll back to older code snapshots (commits) or develop new features without breaking production code.

What is GitHub

- A cloud Git repository & services provider.
- Code management & collaborative development.
- It can handle all the versioning and allows multiple people to collaborate on the same project.
- Repositories support version control capabilities through Git.
- Graphical User Interface, beginner friendly

.git hidden folder

- .git folder is created after you initiate a repository
- .git contains all information required for version control.

GitHub

a central hub for stored code, allowing team members to push and pull changes

An example repository

https://github.com/tytell/CStart

remote

.git

git clone

local

.git

remote

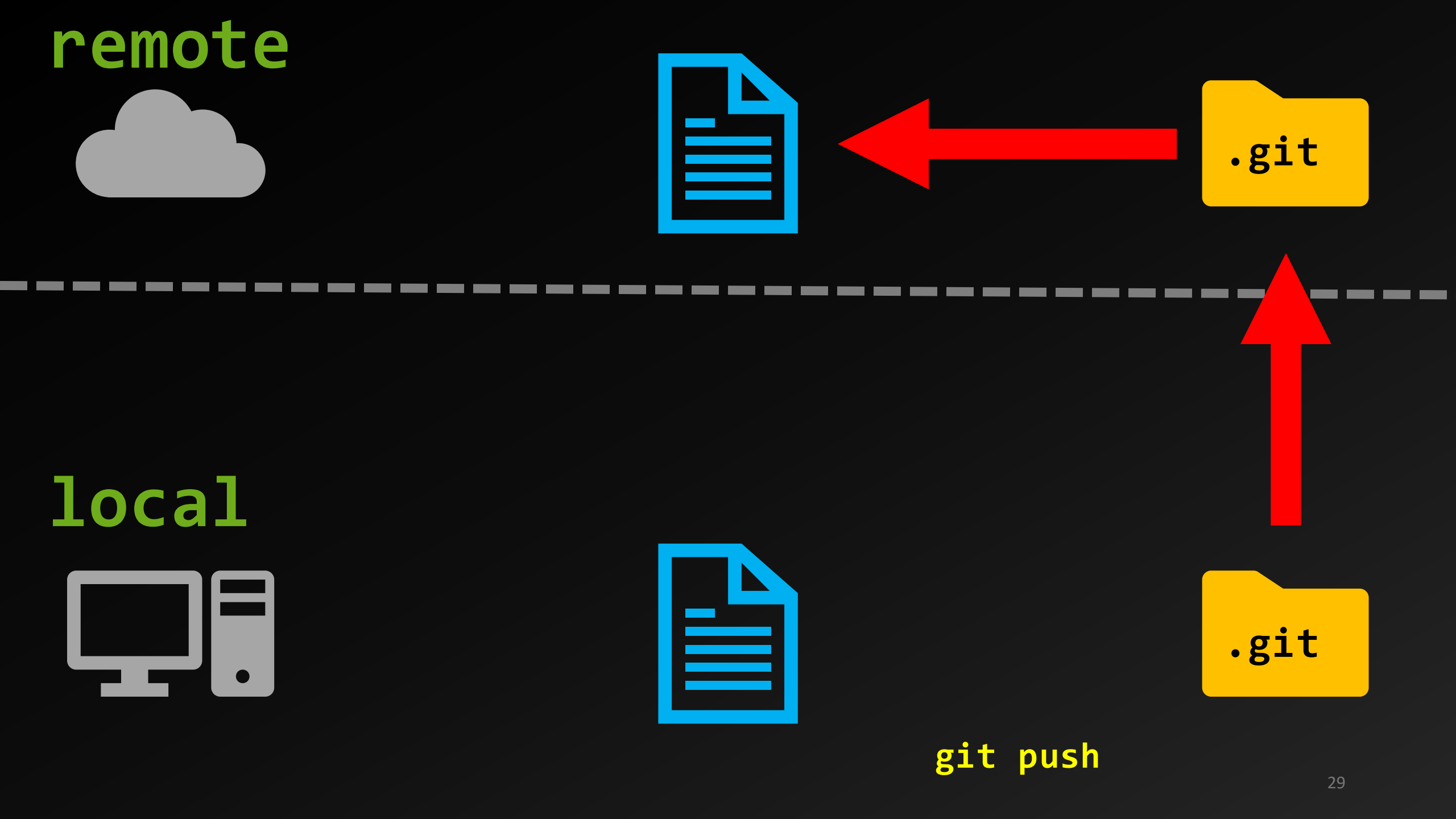
local

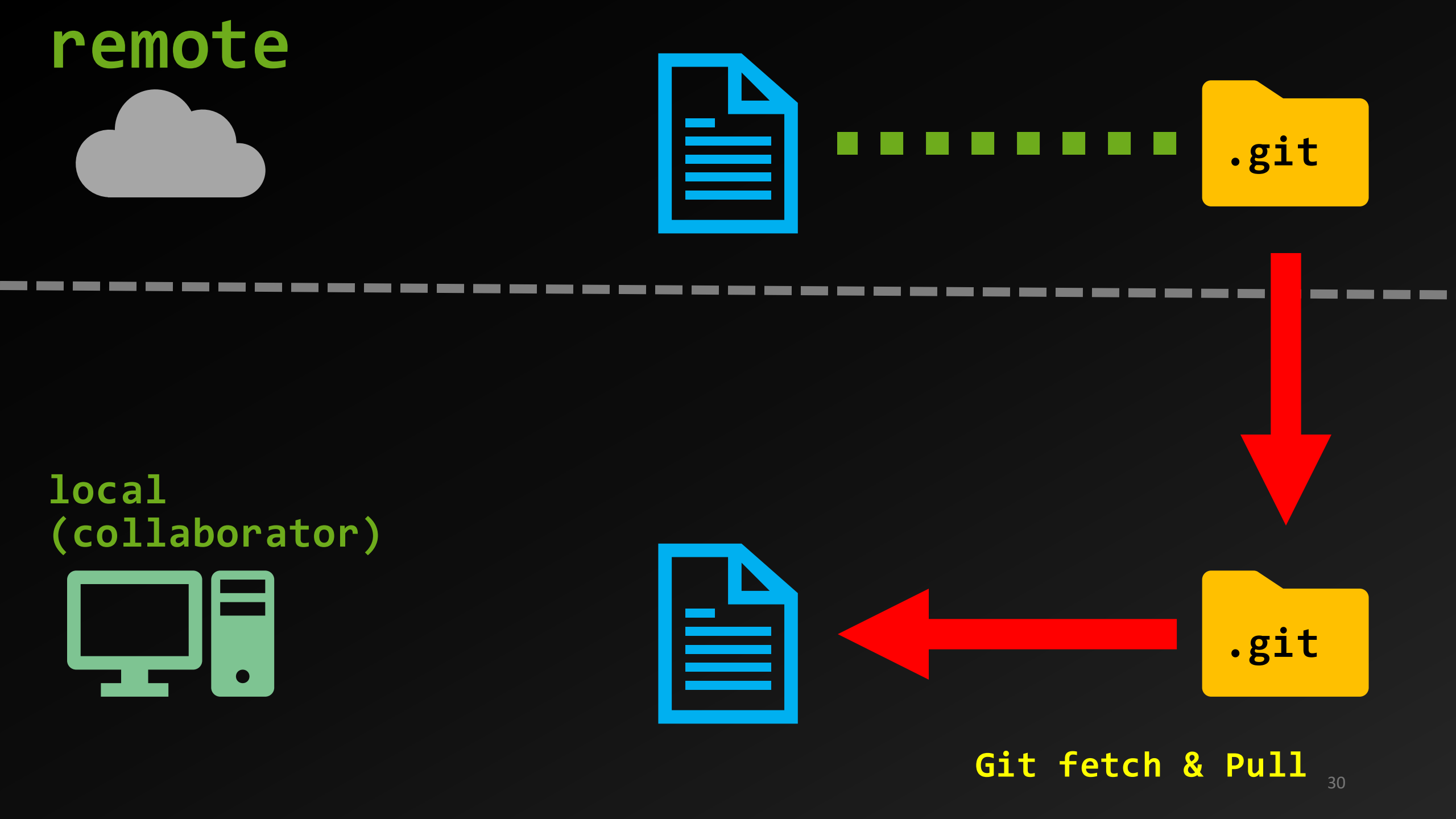
Make changes locally

remote

local

git commit


remote
.git
local
.git
git push


remote
.git
local
(collaborator)
.git
Git fetch & Pull

Git Commit

- A core function in the Git version control system that saves a snapshot of the project’s staged changes, creating a "commit" object in the repository history.
- Each commit includes:
  - Snapshot of Changes
  - Unique Identifier
  - Author Information
  - Timestamp
  - Commit Message

Revert to old version

Commit 1:
Create file.txt

Commit 2:
Delete file.txt

- How to recover file.txt?

A new commit

Commit 1:
Create file.txt

Commit 2:
Delete file.txt

Commit 3:
Revert “Delete file.txt”

Confusions

- **Pull** is a command for updating the local repository to match a remote repository.
- **Push** is a command for updating a remote repository with changes made locally.
- **Pull Request** is a feature allows you to tell others about changes you've pushed to a branch in a repository. (Usage: Merge other branches to main branch; Contribute to open source project.)

Git vs. GitHub Desktop

- Git: a command line tool
  - Git provides more detailed control over all aspects of version control, suitable for complex development workflows.

- GitHub Desktop: a graphical user interface (GUI)
  - GitHub Desktop focuses on simplifying common Git operations, which may limit some advanced functions.

3. Navigating GitHub Desktop

Hands-on demo: https://go.tufts.edu/github2404

4. Resolving Merge Conflicts

How conflicts are generated

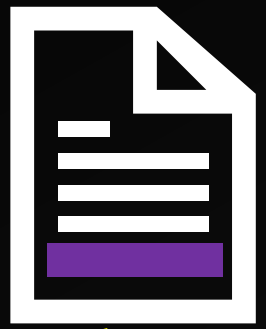
Conflicts

- In services like Box or Dropbox, conflicts arise when two team members simultaneously make changes to the same file.
- Similarly, in a GitHub repository, conflicts occur when two team members modify the same part of a file concurrently.

remote

local

remote



Your collaborator make changes remotely or they make changes on their local copy and commit and push back to GitHub.

local

You make changes locally

remote

local

Commit and push will cause conflicts

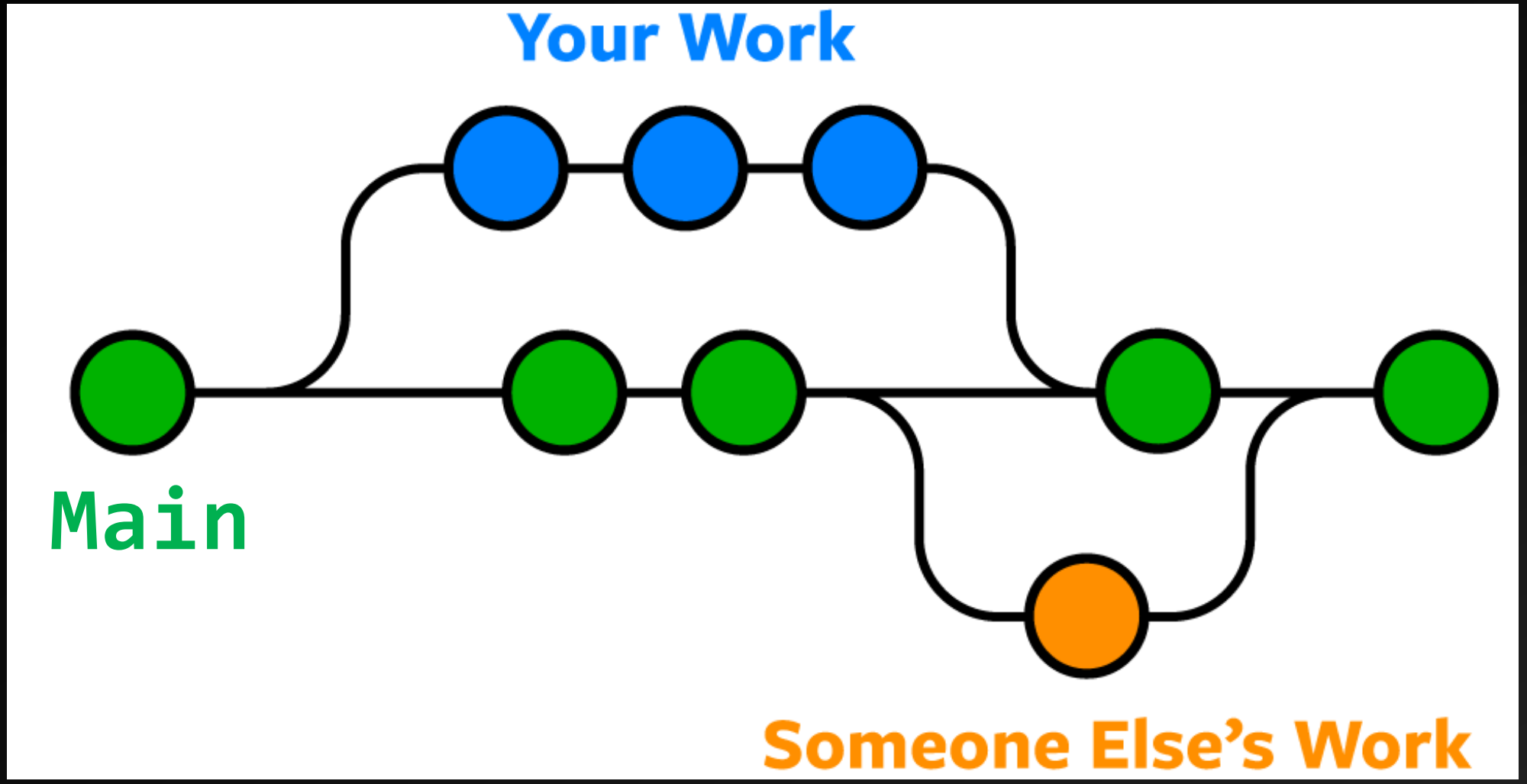
Let’s address conflicts using GitHub Desktop

Hands-on demo: https://go.tufts.edu/github2404

5. Understanding Branch Workflow

Introduction to Branching

- **Definition:** A branch in version control is a separate line of development that diverges from the main line (often called "master" or "main").
- **Purpose:** Allows developers to work simultaneously on different features or fixes without disturbing the stable version of the project.
- **Isolation:** Changes in one branch don't affect others, making it safer to experiment.
- **Collaboration:** Multiple people can work on different features simultaneously without interference.

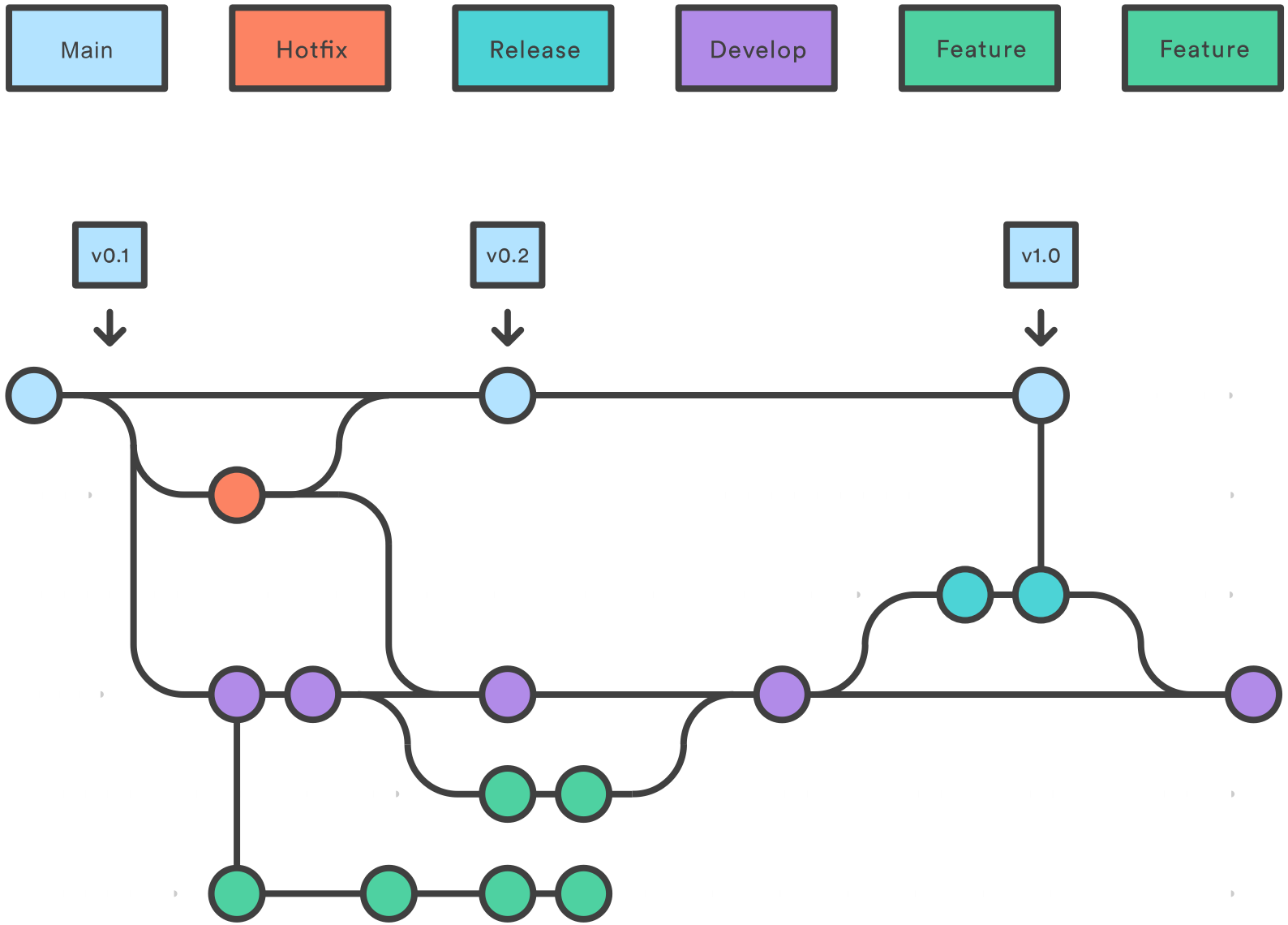

Your Work
Main
Someone Else's Work

Branch Workflow Steps

1. Create a Branch
2. Add Commits
3. Open a Pull Request
4. Review and Merge
5. Delete the Branch

Best Practices

- Keep Branches Short-Lived
- Regularly Sync with Main Branch


Main
Hotfix
Release
Develop
Feature
Feature
v0.1
v0.2
v1.0

How conflicts are generated when merging branches

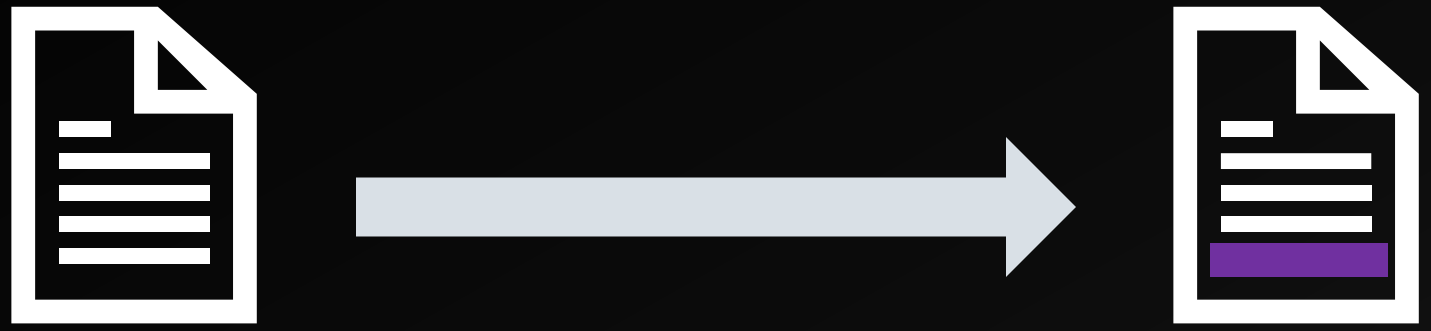
Main branch

Test branch

Main branch

Test branch

Main branch

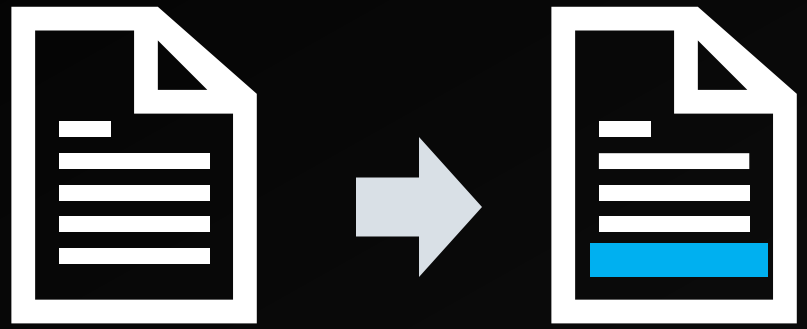


Test branch

Merge back to main branch

No Conflict

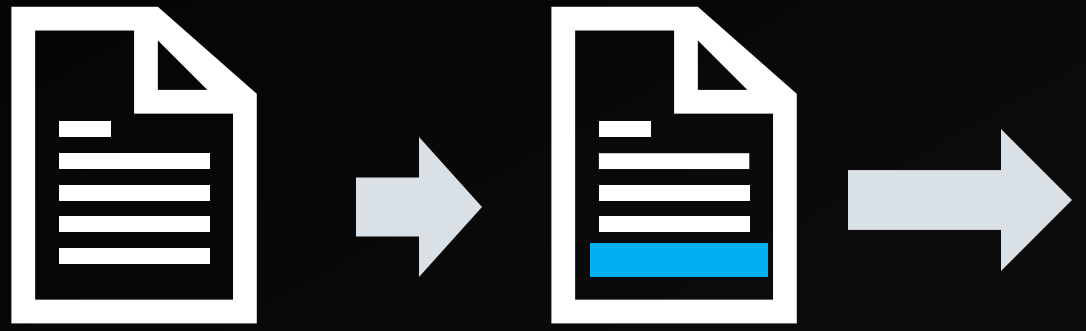
Main branch



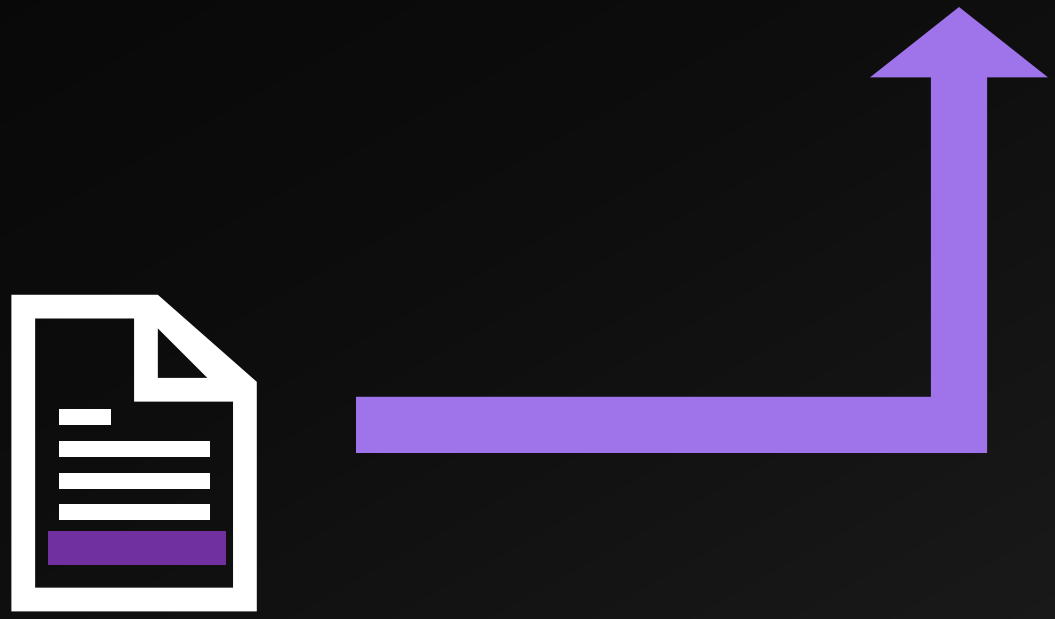
Test branch

With Conflict

Main branch

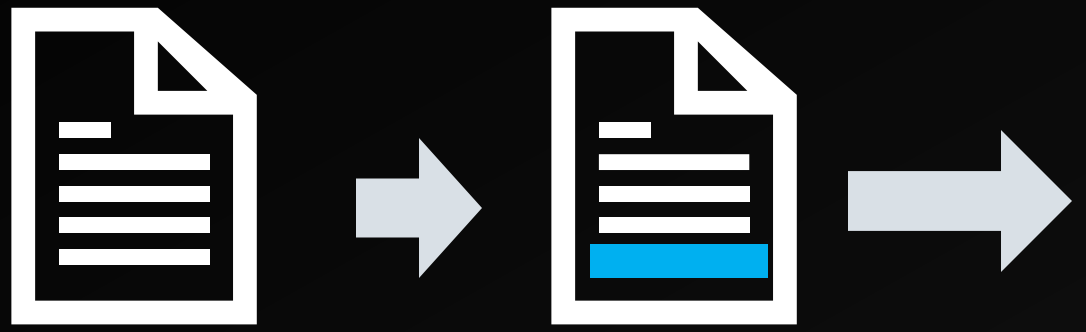


Test branch

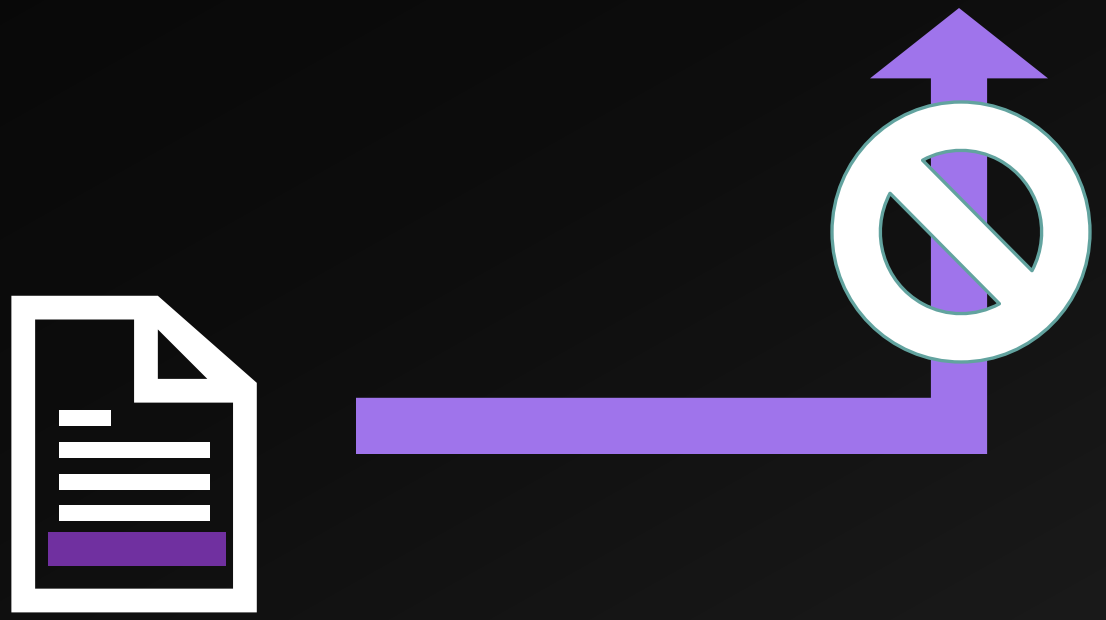


With Conflict

Main branch



Test branch



With Conflict

Resolve the conflict and merge again

Let's practice with GitHub Desktop

Hands-on demo: https://go.tufts.edu/github2404

6. Best practices for GitHub Usage

Best practices

- Sync frequently to make your changes available to others. Frequently: Fetch&pull before making any local changes.
- Small, Frequent Commits.
- Push Regularly: Push your commits
- Branch Strategically: Use branches to manage features, bug fixes, and experiments separately from the main codebase.
- Communicate Regularly.
- Document Changes: Update README.
- Do not store large files in GitHub. It has limited storage.

Good commit

- Single Focus: Each commit should represent a single logical change.
- Small Size: Smaller commits are easier to understand and less likely to introduce complex merge conflicts.
- Always write good commit message.

Good commit message

- Concise, specific.
  - “Add user authentication system”
  - Avoid “Update of file.txt”, “Fixed it”
- Detailed explanation including what, why, and how.
- Not too long, not too short. ~50 characters.
- References to related issue or pull requests: “See also #46”

More about best practices

- https://docs.github.ncsu.edu/github-best-practices/
- https://docs.github.com/en/repositories/creating-and-managing-repositories/best-practices-for-repositories
- https://github.com/orgs/community/discussions/39082
- https://dangitgit.com/

Recommended tools

https://code.visualstudio.com/

Popular tools to compare code differences

- KDiff3: https://kdiff3.sourceforge.net/
- Beyond Compare: https://www.scootersoftware.com/
- WinMerge: https://winmerge.org/?lang=en
- Code Compare: https://www.devart.com/codecompare/

More about .gitignore

- https://git-scm.com/docs/gitignore
- https://www.atlassian.com/git/tutorials/saving-changes/gitignore
- https://www.youtube.com/watch?v=4a2ZVSzMMq8

Shirley Li, Bioinformatician, TTS

xue.li37@tufts.edu