

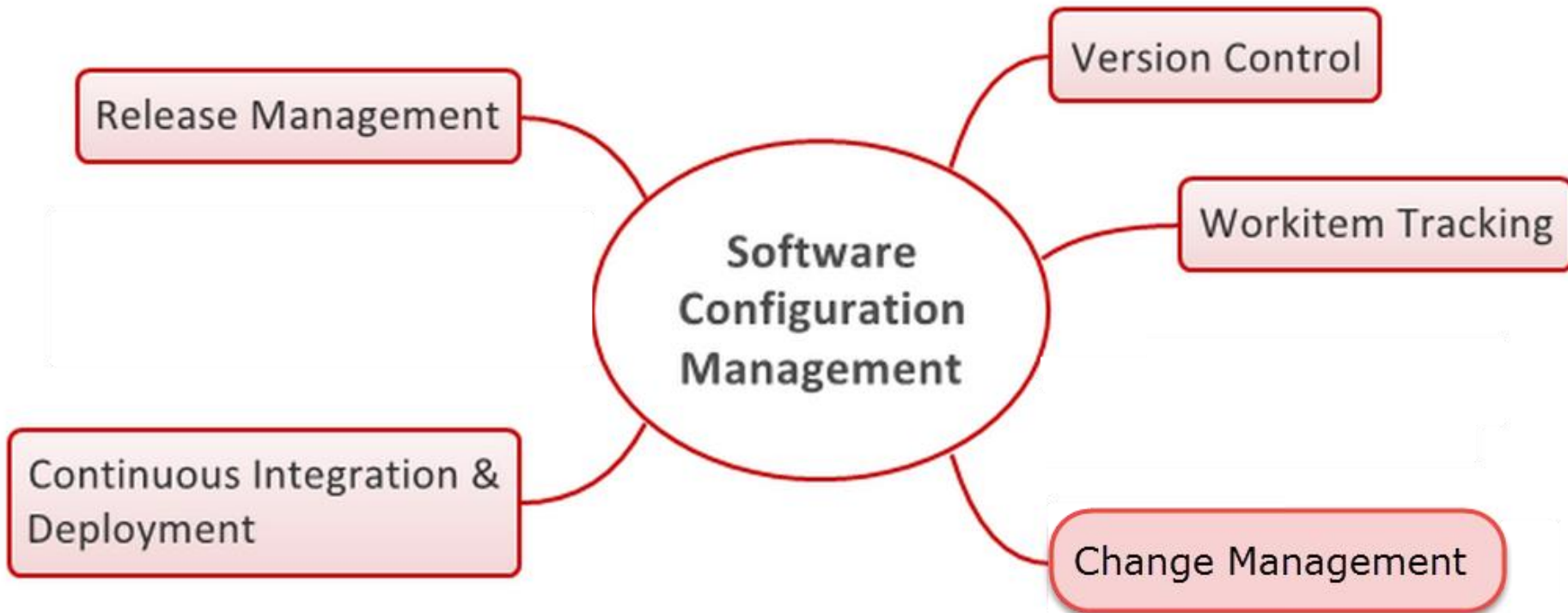
Git and Github

Cloud-based Software Configuration Management
System

Software Configuration Management (SCM)

- Software Configuration Management
 - Techniques, practices and tools to track and **manage changes** throughout the software life cycle
 - Defines the process of change
 - Keeps track of what is happening in the project:
 - Which changes has been made
 - Who did those changes, and
 - Why

SCM Aspects

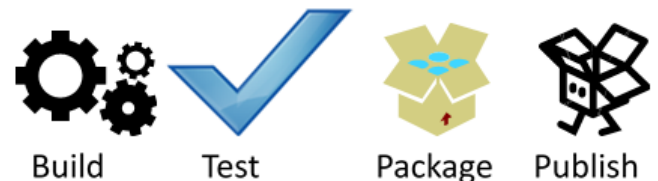
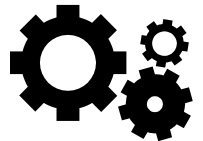


Requirements for SCM

- Repository: shared DB for artifacts with controlled access to prevent overwrites.
- Version management: Maintain history of changes made to each artifact; provide ability to see how version was created.
- Work Item Tracker: To manage tasks, issues and bugs.
- Product build and deployment: Automated build and deployment of the product from artifacts in repository.

SCM Tools

- Version control
 - git, github, CVS, Subversion
- Bug tracking
 - Bugzilla, Mantis Bugtracker, Rational ClearQuest
- Automated Build
 - Maven, Ant
- Continuous Integration (build, test and deploy)
 - Jenkins



Version Control



Version Control

- Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.
- Why?
 - Revert files back to a previous state
 - Compare changes over time
 - See who last modified something
 - Generally, if you screw things up or lose files, you can easily recover

Versioning Models

- **Lock-Modify-Unlock:**
 - Only one user works on a given file at a time → no conflicts
 - Example: Visual SourceSafe, Team Foundation Server (TFS)
- **Copy-Modify-Merge:**
 - Users make parallel changes to their own working copies
 - The parallel changes are merged and the final version emerges
 - Examples: Git, CVS, Subversion

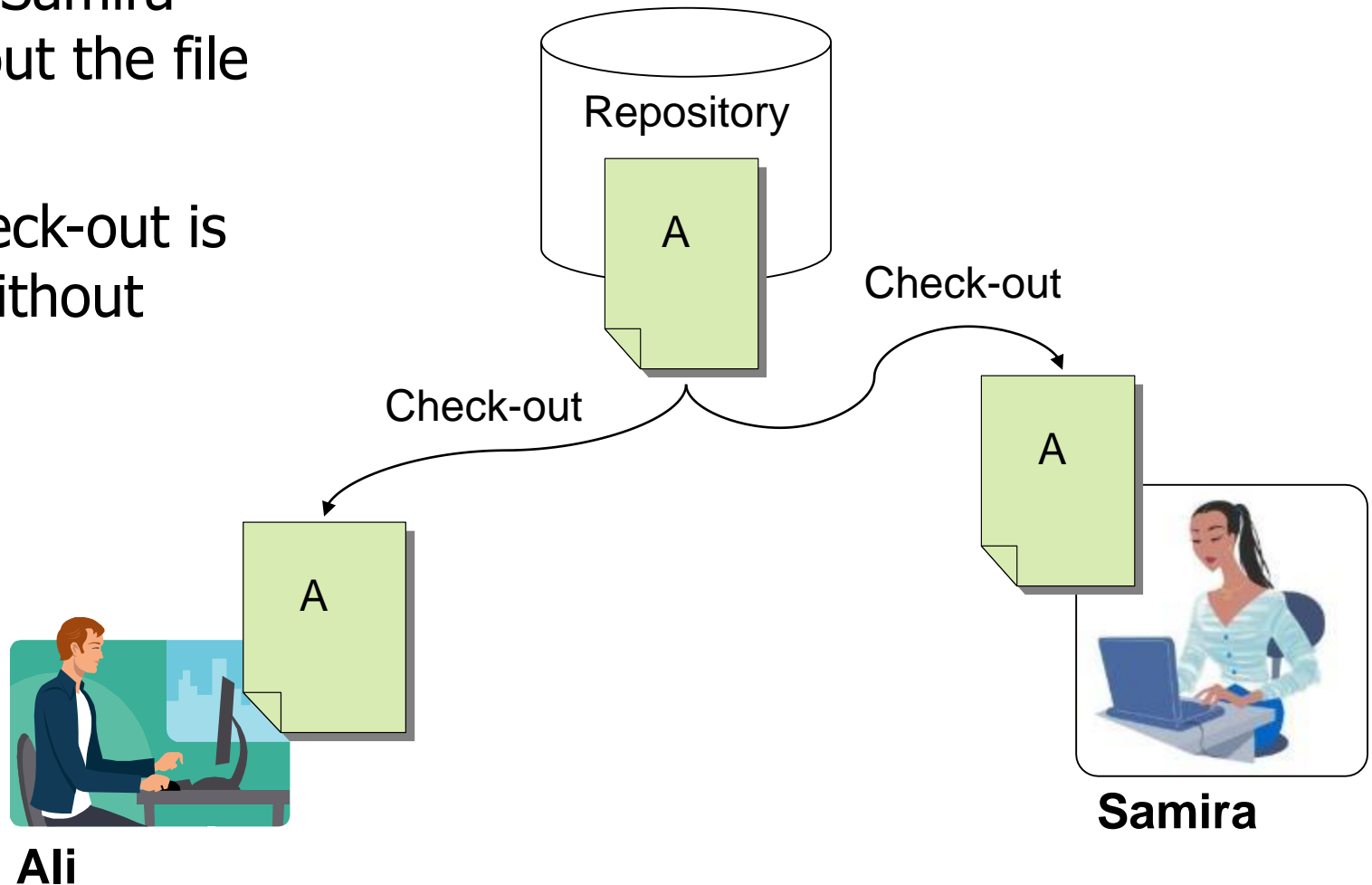
Merging Problems

- If a given file is concurrently modified it is necessary to merge the changes
 - Merging is hard!
 - It is not always possible to do it automatically
- Responsibility and coordination between the developers is needed
 - Commit as fast as you can
 - Do not commit code that does not compile or blocks the work of the others
 - Add comments on commit

The Copy-Modify-Merge Model (1)

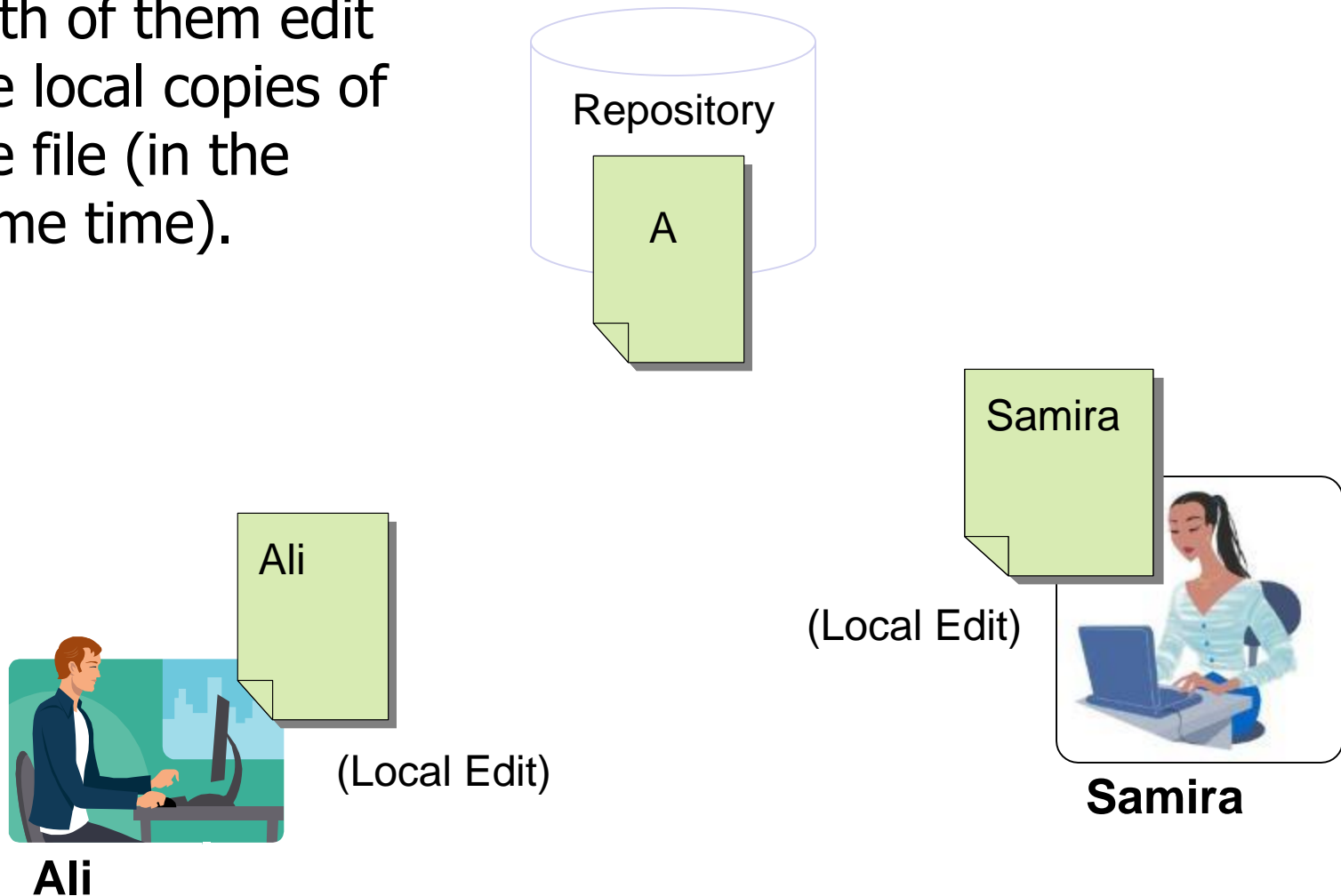
Ali and Samira
check-out the file
A.

The check-out is
done without
locking.



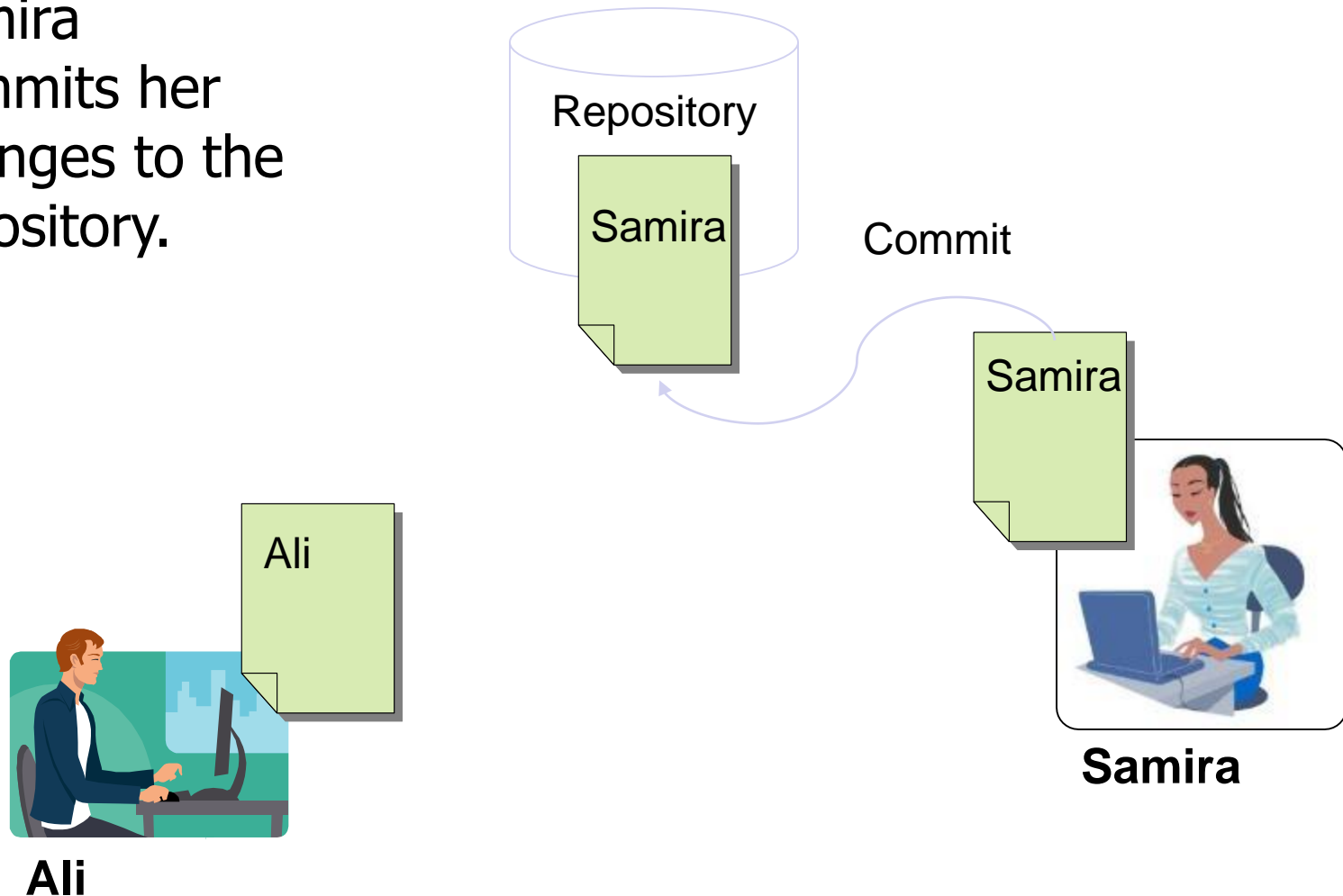
The Copy-Modify-Merge Model (2)

Both of them edit the local copies of the file (in the same time).



The Copy-Modify-Merge Model (3)

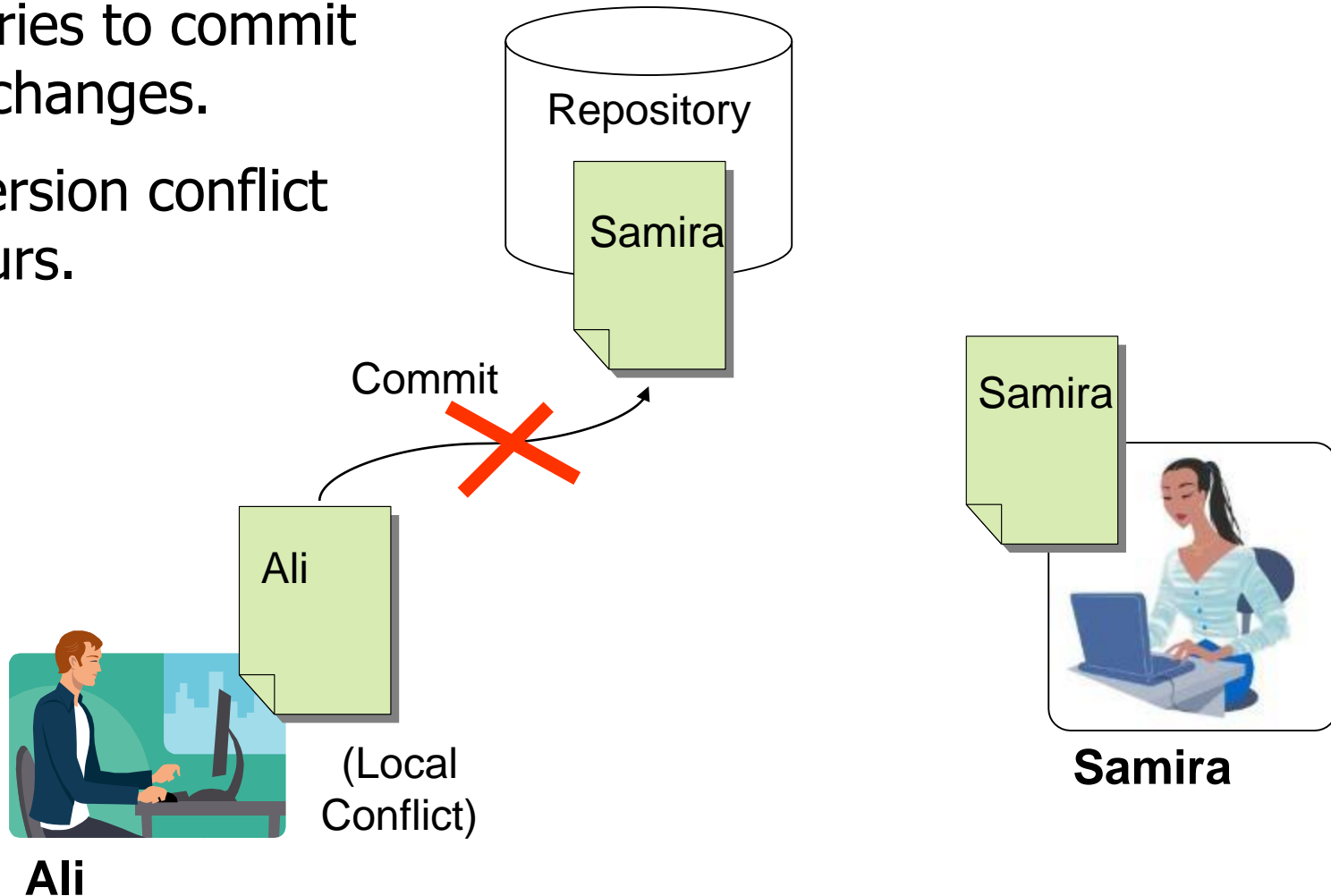
Samira
commits her
changes to the
repository.



The Copy-Modify-Merge Model (4)

Ali tries to commit his changes.

A version conflict occurs.

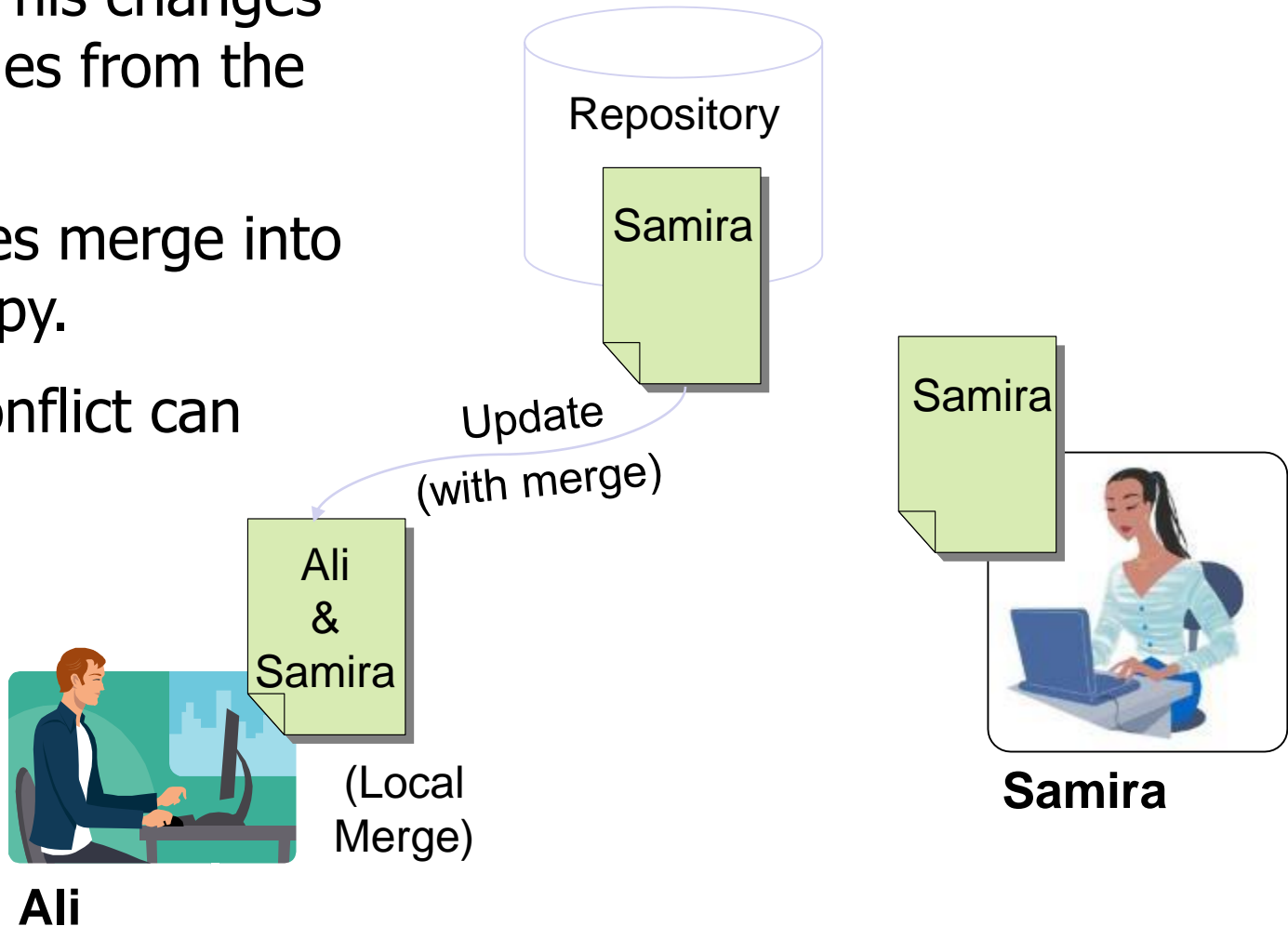


The Copy-Modify-Merge Model (5)

Ali updates his changes with the ones from the repository.

The changes merge into his local copy.

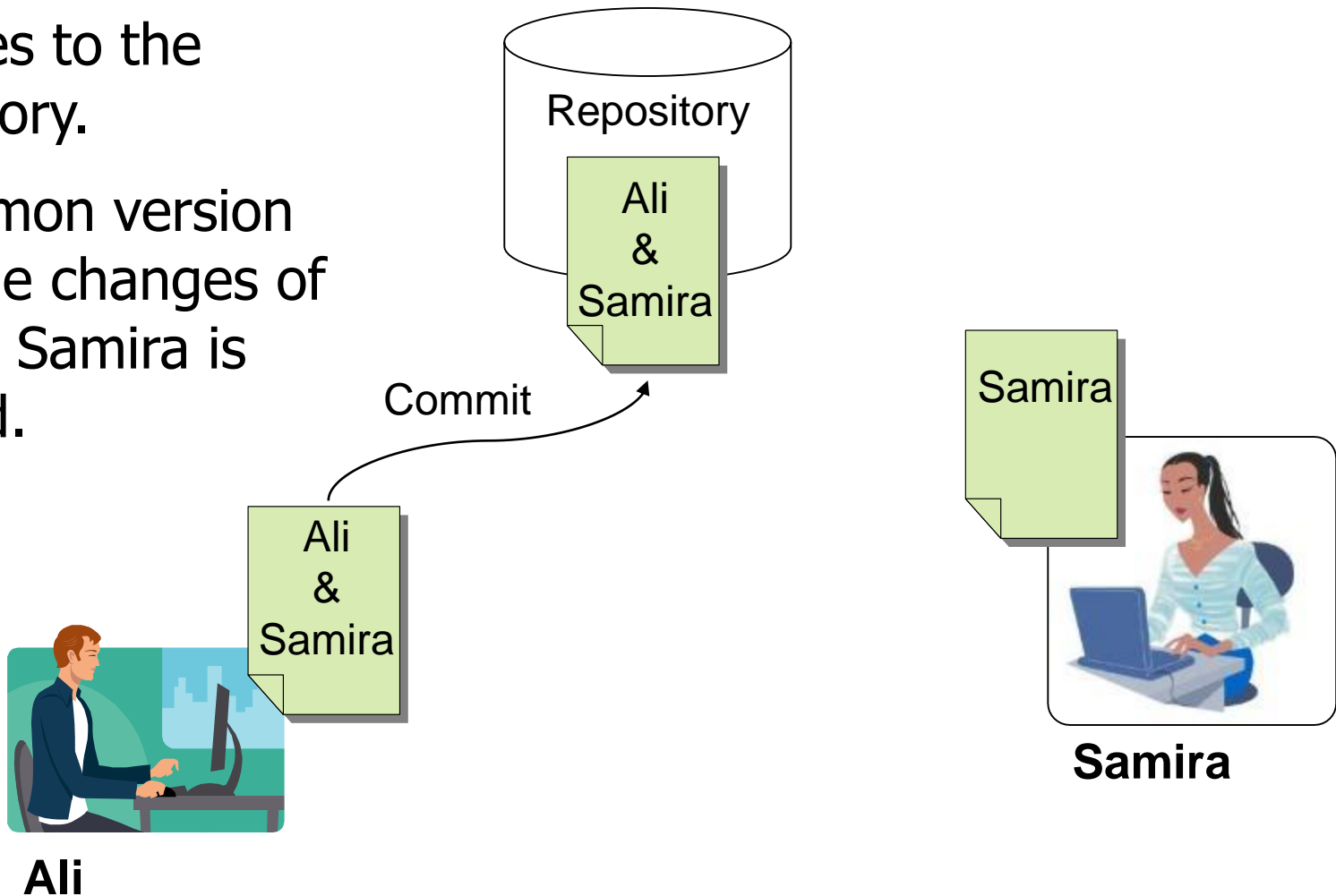
A merge conflict can occur.



The Copy-Modify-Merge Model (6)

Ali commits the changes to the repository.

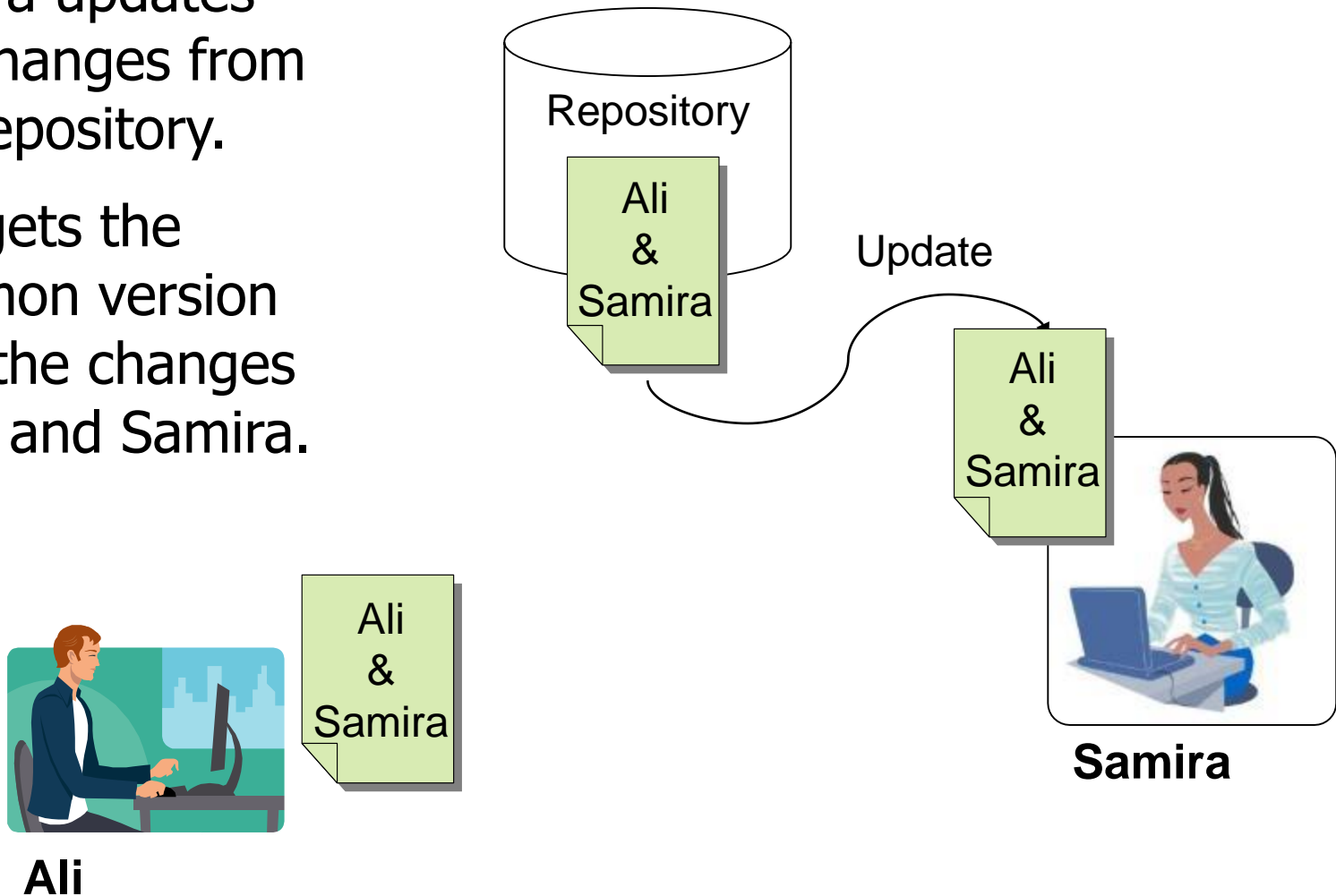
A common version with the changes of Ali and Samira is pushed.



The Copy-Modify-Merge Model (7)

Samira updates
the changes from
the repository.

She gets the
common version
with the changes
of Ali and Samira.

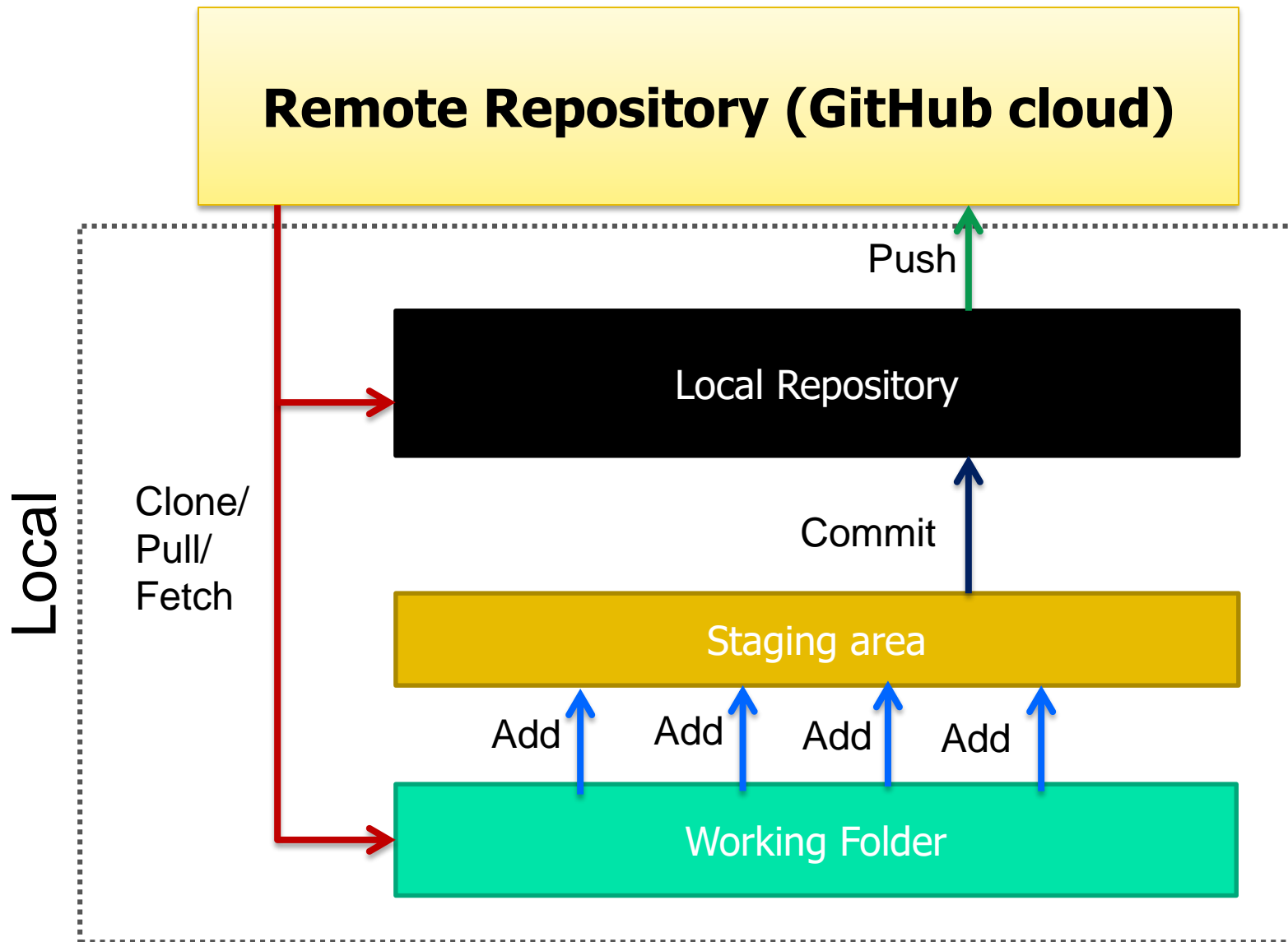


Git and Github

Github

- Github is a distributed source control management system
- It also provides **access control** and several collaboration features such as **wikis**, **task management**, and **bug tracking**

Architecture & Terminology

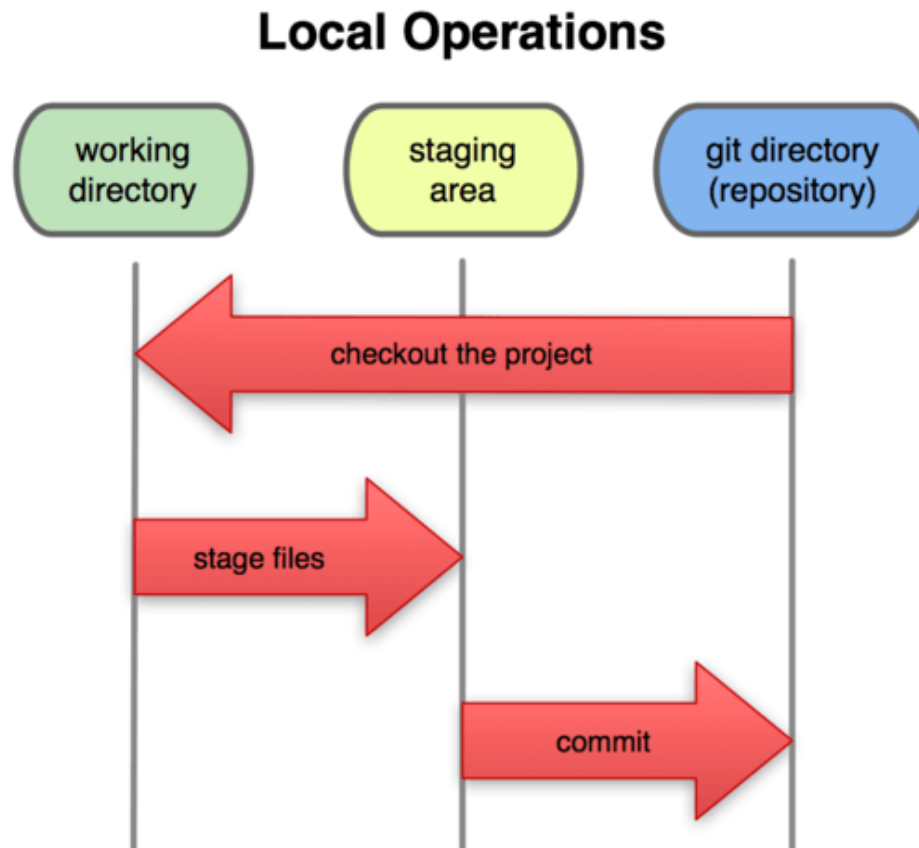


Git Basics

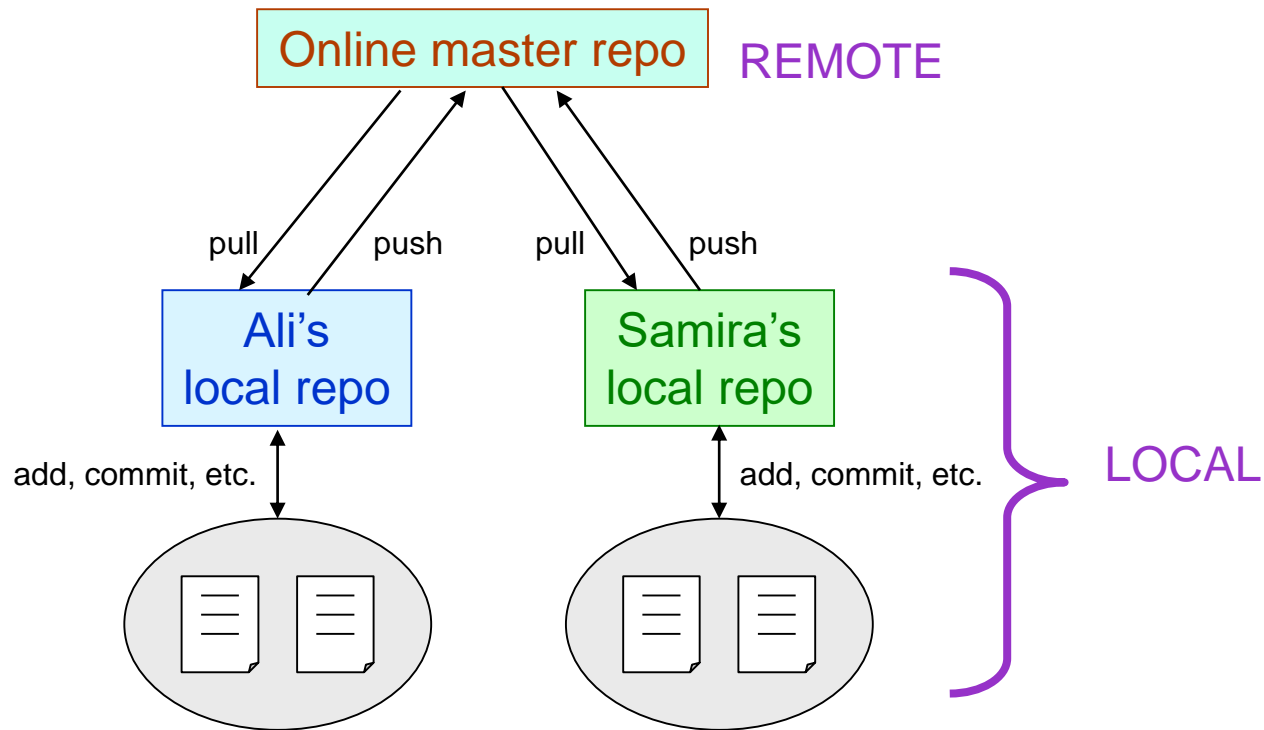
- The 4 States of a file
 - Modified
 - File has changed but not committed
 - Staged
 - Marked to go to next commit snapshot
 - Committed
 - Safely stored in local database
 - Untracked!
 - Newly added or removed files

Git Basics (Cont.)

- Three Main Section of a Git Project



Local and Remote Repositories



GitHub: Create Local Repository

- Each team member creates local repository that is a **clone** of the master repository.
 - Log into your personal GitHub account
 - Navigate to the team repository
- Copy the **URL to the team repository**.

HTTPS clone URL

`https://github.com/c`



GitHub: Create Local Repository, *cont'd*

- **cd** to the directory where you want the local repository to reside on your local machine.
- Enter the git command

```
git clone URL
```

- where *URL* is the URL from the clipboard
 - Example:

```
git clone https://github.com/cmpt561f15/Examples.git
```


Git: Make Local Changes

- Get the status of files in your local repository:

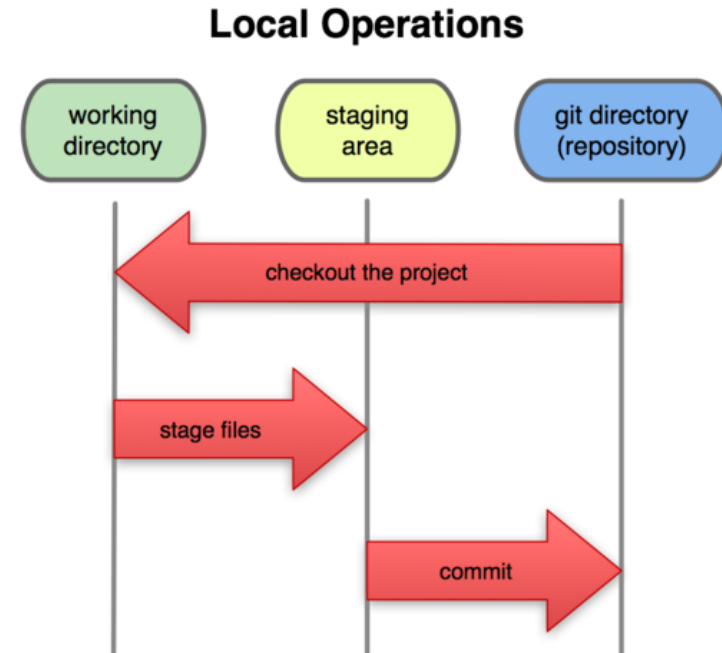
```
git status
```

- After you've created new files on your working directory, first **add** them to the **local staging area**:

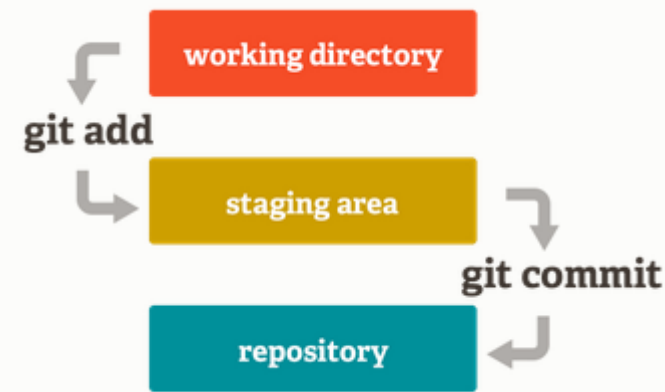
```
git add myfile1 myfile2
```

- Commit** your staged/modified files to the **local repository**:

```
git commit -m "commit message"
```



Git Basic Commands Summary



`git init` //initializes a new git repo

`git add filename` //adds file to the local staging area

`git diff` //prints difference made in files

`git commit -m "Message here "` //save changes to local repository

`git status` //prints status of current repository

`git log` //history

`git push [options] origin branch_name` //updates a remote repository with the changes made locally

Git: Local Operations

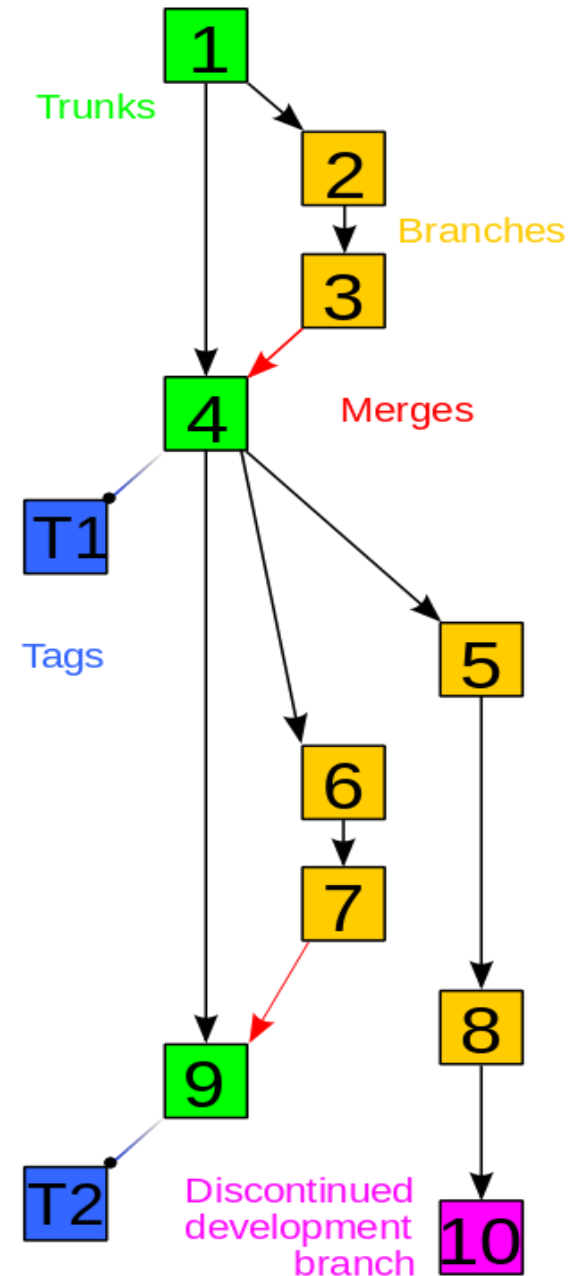
- Each developer has a **complete local copy of the repository** on his or her workstation
- **Local operations**
 - add
 - commit
 - check in
 - check out
 - status
 - differences
 - etc.

Git: Remote Operations

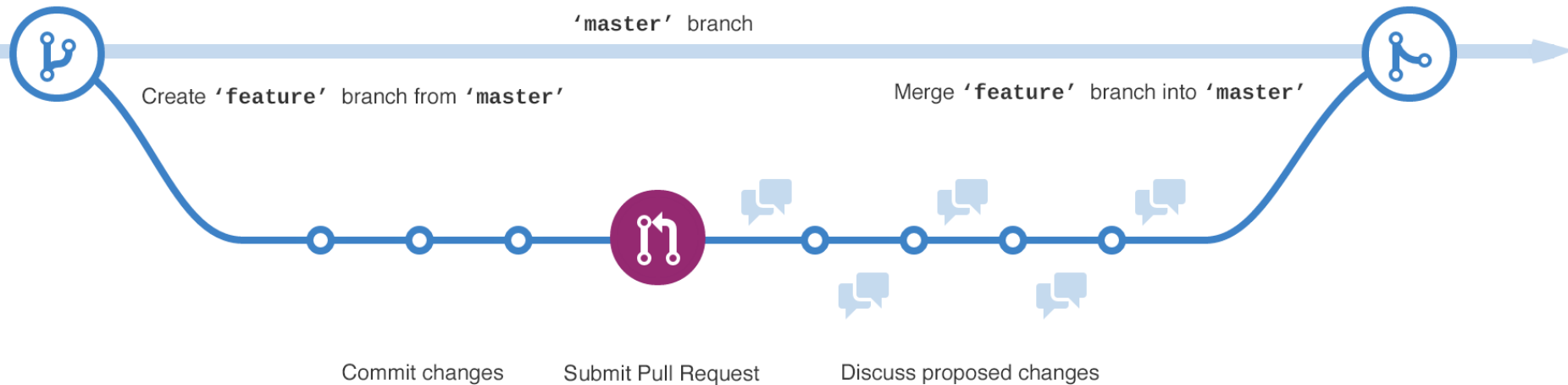
- Do remote operations **only when necessary** to the master repository on the server.
 - **Clone** the master repository into a local repository
 - **Push** (and merge) a local repository up to the master repository
 - If another team member had pushed the same files since you last obtained them from the master repository, or added new files, you'll have to **pull** down the changed or new files in order to merge them into your local repository before you can push
 - **Pull** (and merge) files from the master repository down to a local repository

History Tree

- Trunk is in green is the master branch that has the production version
- Branches are in yellow
- A need of branching arises in the following situations:
 - Developing a new feature or fixing a bug
 - **Variant:** functionally equivalent versions, but designed for different settings, e.g. hardware and software



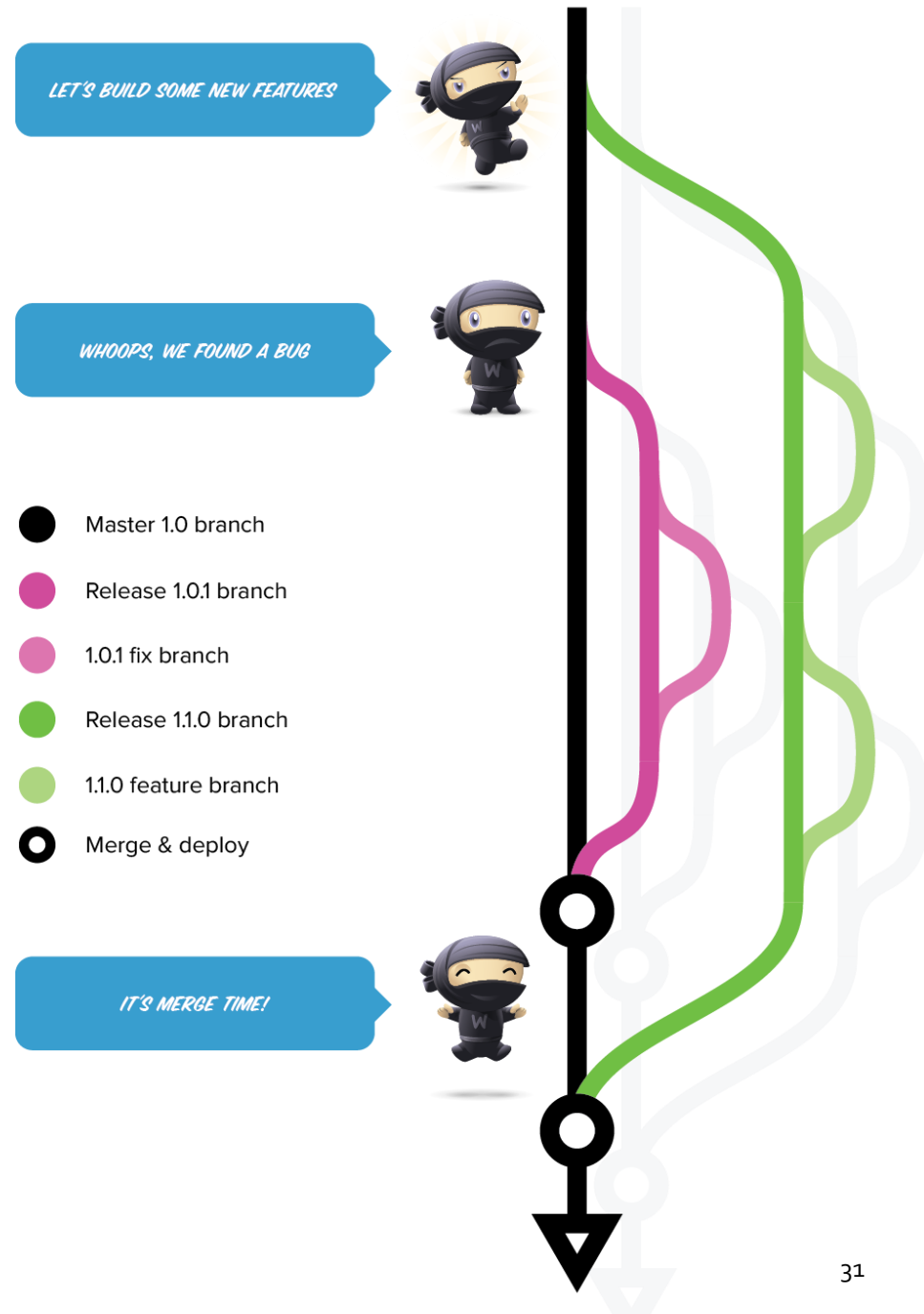
Branches using GitHub



- Developers use branches for keeping bug fixes and feature work **separate** from the **master** (production) branch. When a feature or fix is ready, the branch is merged into master.
- Before merge you may make a **pull request**, to start a discussion about commits (code review) and get feedback

Branches

- **Isolating** new development from finished work
- New development (new features, non-emergency bug fixes) are built in **feature branches**.
 - They are only **merged** back into master branch when ready for release



Resources

- Github foundation short videos @

<https://www.youtube.com/playlist?list=PLologMOBetEHhfGgvJzVCTiDYcbhAiEqL>

- Hello world github branching lab:

<https://guides.github.com/activities/hello-world/>
(without commands)

- Outstanding tutorials

<http://jlord.us/git-it/index.html>