CMPS 411

Software Configuration Management (SCM)

Please read Chapter 22

Dr. Abdelkarim Erradi

Dept. of Computer Science & Engineering

QU

Outline

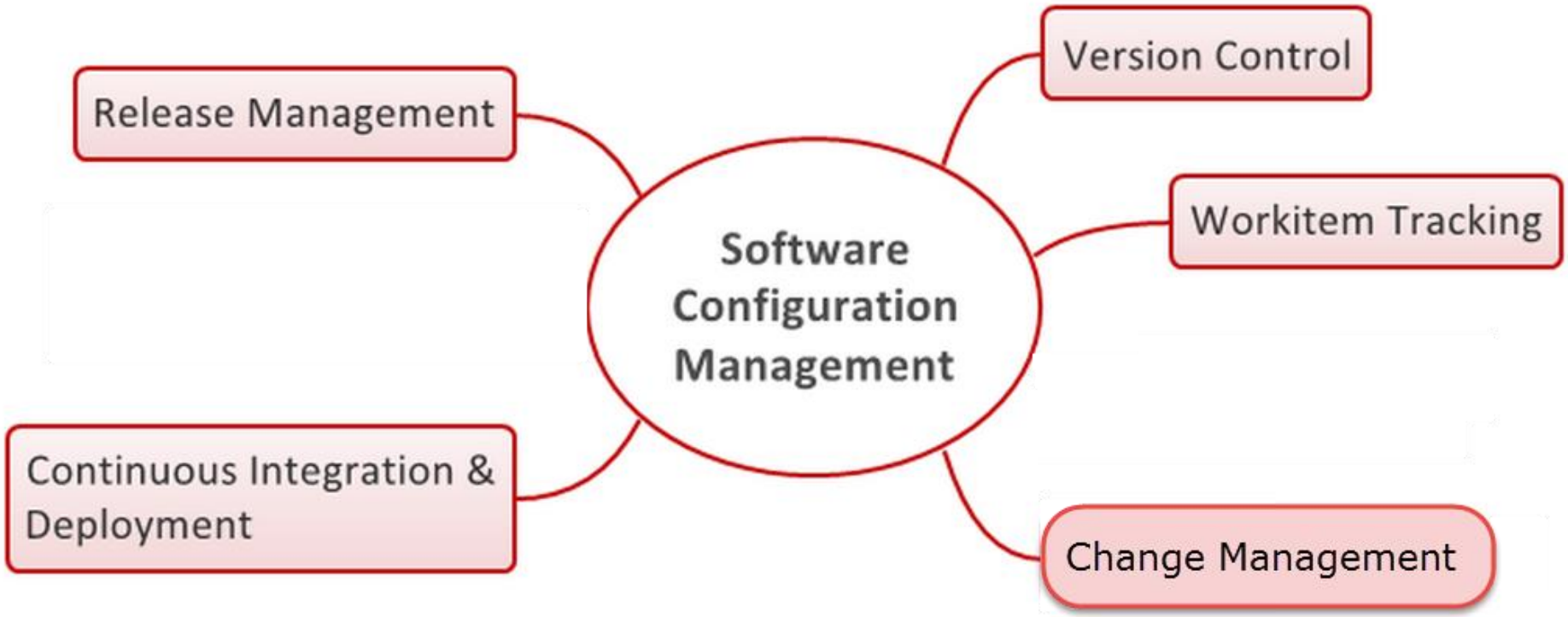
1. Software Configuration Management (SCM)
2. Version Control
3. Versioning Models
   - Lock-Modify-Unlock
   - Copy-Modify-Merge
4. Project Hosting Sites
5. Git and Github

Software Configuration Management (SCM)

Software Configuration Management (SCM)

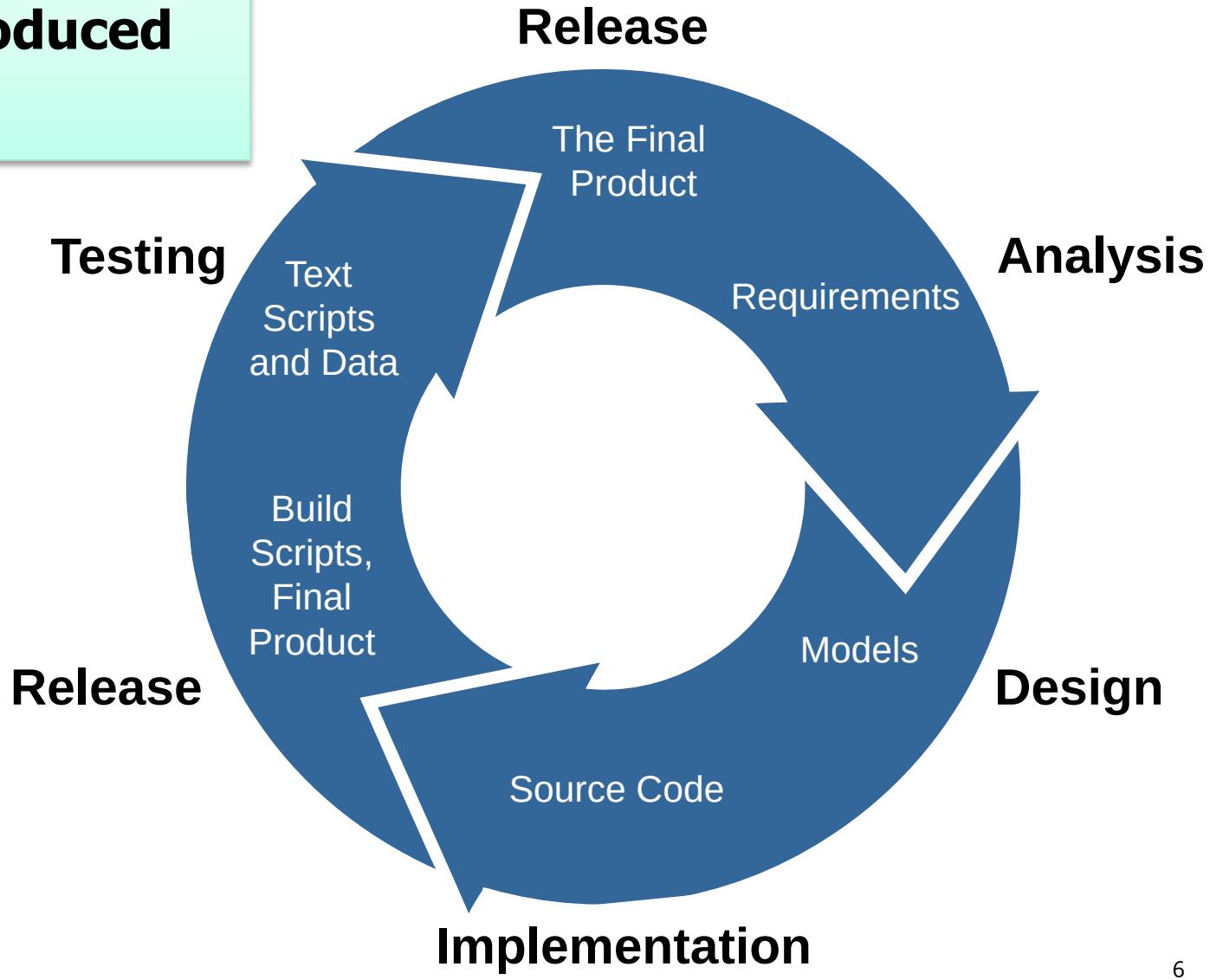
- Software Configuration Management
  - Techniques, practices and tools to track and **manage changes** throughout the software life cycle
  - Defines the process of change
  - Keeps track of what is happening in the project:
    - Which changes has been made
    - Who did those changes, and
    - Why

SCM Aspects

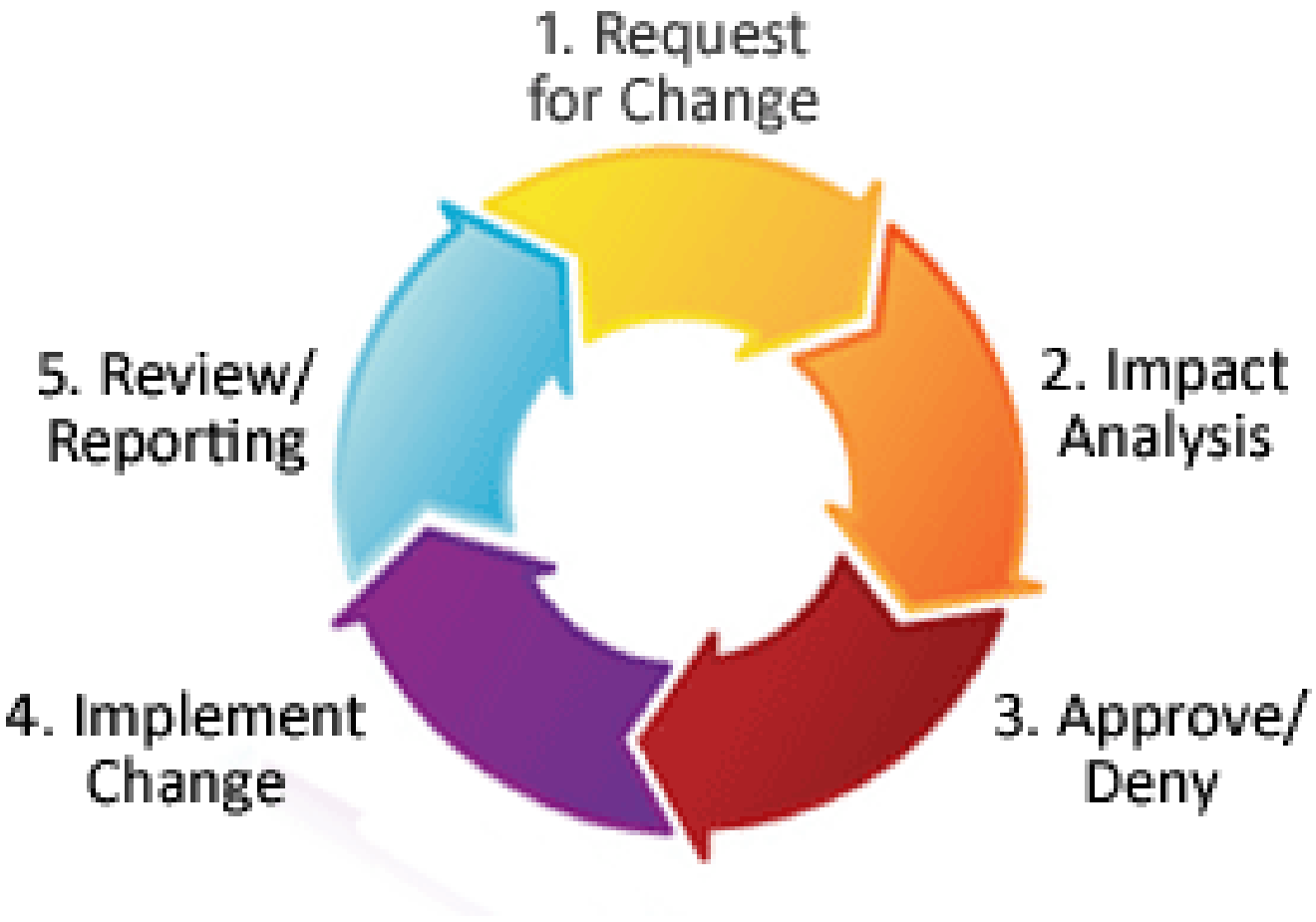

Release Management
Version Control
Software Configuration Management
Workitem Tracking
Continuous Integration & Deployment
Change Management

SCM and Software Development Lifecycle (SDLS)

SCM manages **changes** to **artifacts produced during SDLS**



Change Management Process



Factors in Change Analysis

- The consequences of not making the change
- The benefits of the change
- The number of users affected by the change
- The costs of making the change
- The product release cycle

Software Configuration Item (SCI)

- Definition: Artifacts that are created as part of the software engineering process.
- Examples:
  - Software Requirements Specification
  - So*f*tware Project Plan
  - Models
  - Design document
  - Source code
  - Test suite
  - Build and deployment scripts

Baseline

- Baseline = software artifact that:
  - has been ***formally reviewed*** and ***agreed upon***,
  - serves as the basis for further development, and
  - can be ***changed only through formal change*** control procedures.
- One "*official version*" at any point in time
- Helps control change without impeding justifiable change.

Requirements for SCM

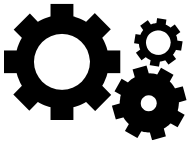
- Repository: shared DB for artifacts with controlled access to prevent overwrites.
- Version management: Maintain history of changes made to each artifact; provide ability to see how version was created.
- Work Item Tracker: To manage tasks, issues and bugs.
- Product build and deployment: Automated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

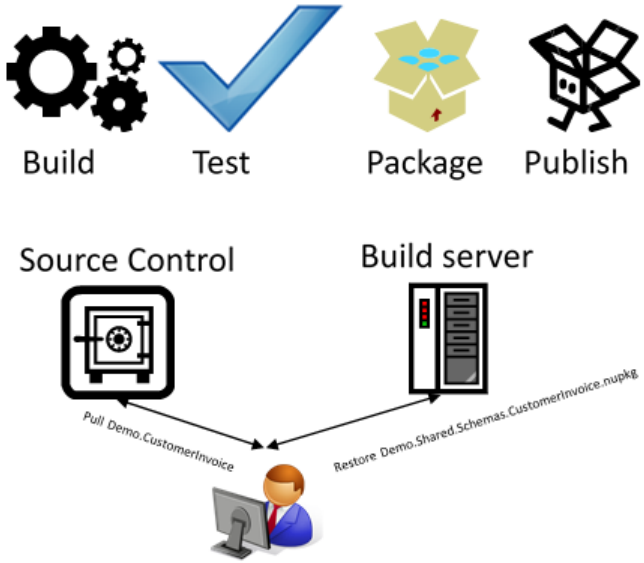


- Continuous Integration (build, test and deploy)
  - Jenkins

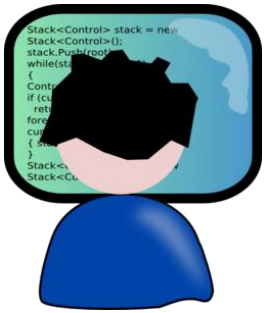
Continuous Integration

- Continuous Integration is a software development practice where members of a team **integrate their work** (merge their local copies) frequently
  - ***Build Server*** integrates and compiles the project, runs all tests, and if successful, deploys the build to a testing environment

- e.g., https://jenkins-ci.org/



Typical CI Process



1. checks in code

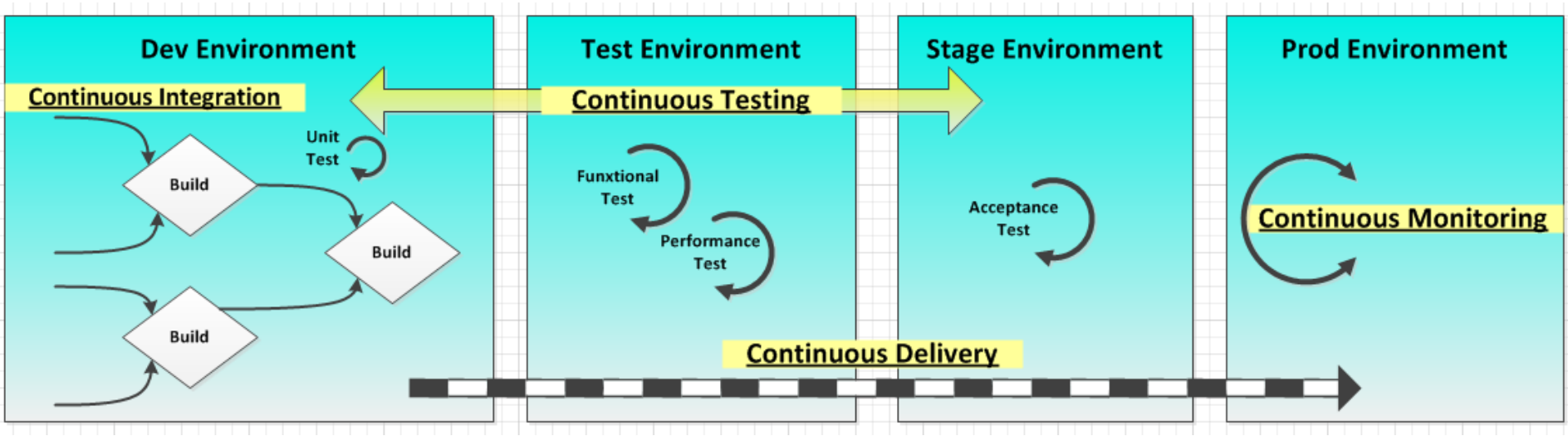
Source Control Management

Build System

2. compiles code
3. runs unit tests
4. creates install package

5. deploys install package to target environment
6. runs integration/regression tests
7. reports progress of each step

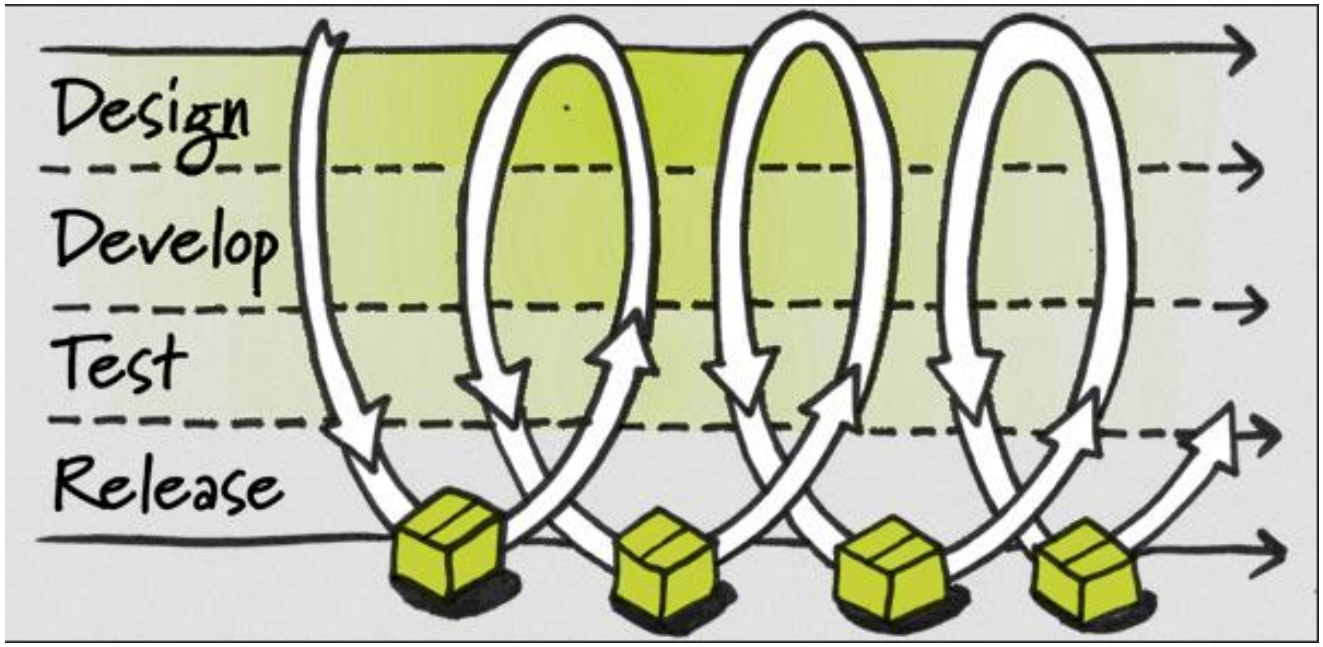
Continuous Integration



http://bit.ly/PRQ9dQ

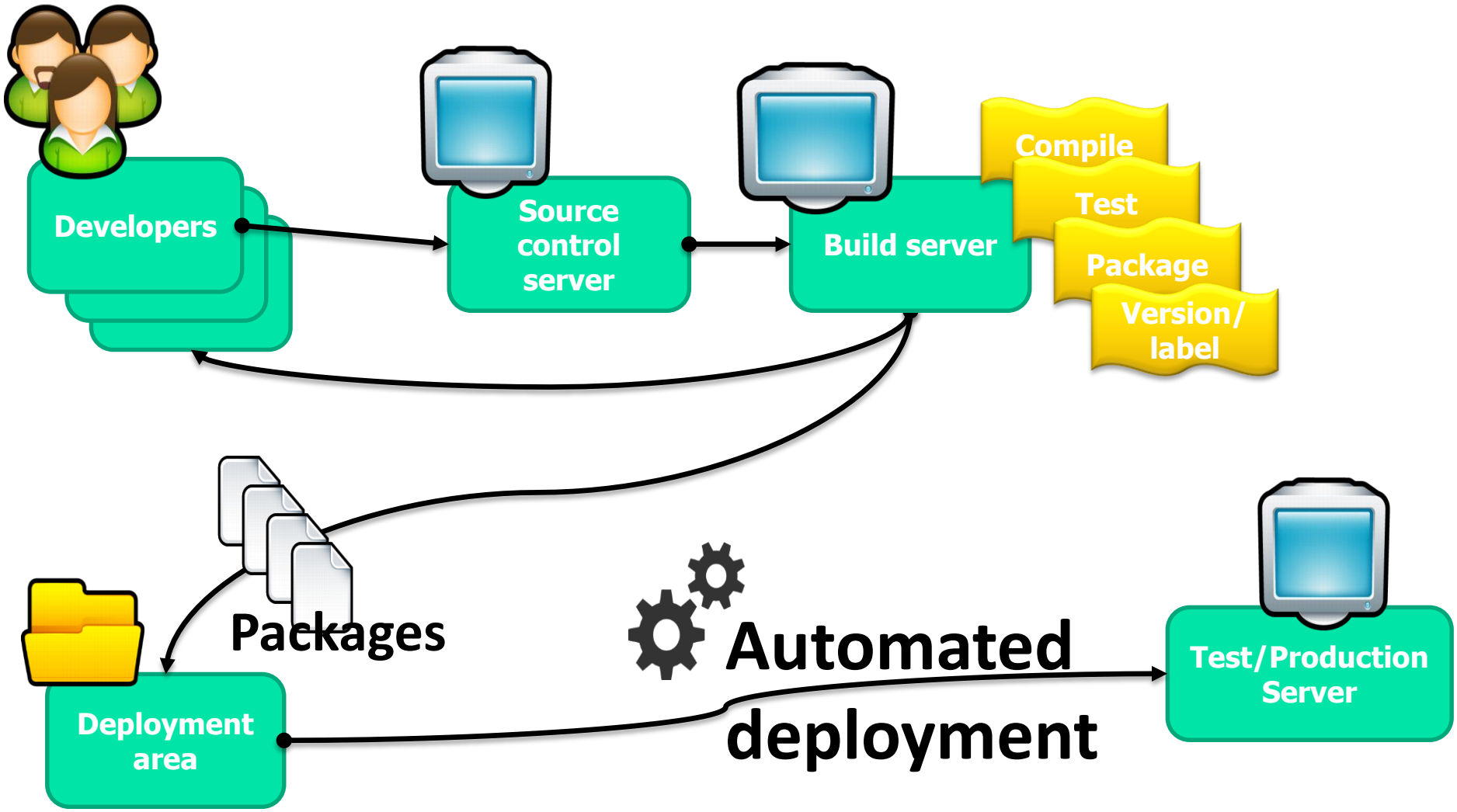
Continuous Delivery

- Build software that is always ready to be deployed into production

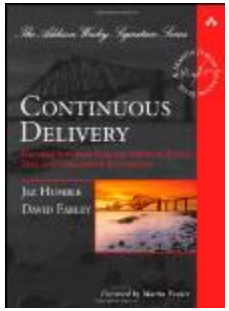


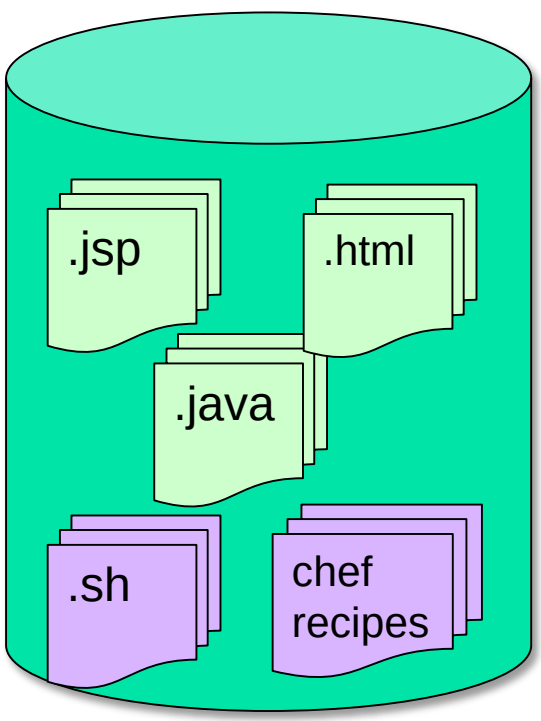
Continuous Delivery Assembly Line Metaphor
https://www.youtube.com/watch?v=SIaVsG7m8n4

What's Continuous Integration/delivery/deployment really?



Delivery Pipeline



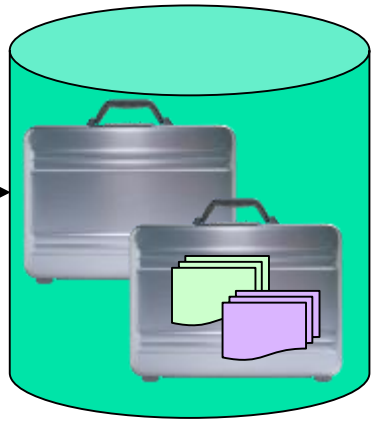


Source Artifacts

Source Control Management

Build, Package, & Unit Test Application Binaries & Platform Configuration

Automate



Deployable Artifacts

Library

Deploy

Automate

Test/Production Environment

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

What is revision control?

WITHOUT

- If a team creates a bug, it could affect how your code too
- You could loose old code or features that were removed
- Anyone could add bugs/features to a project and no one would know

WITH

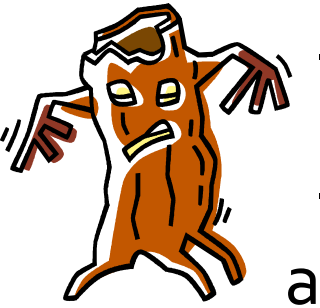
- You are isolated on your own branch, so you know who did it
- You can go back and see all old version of your project
- Someone has to approve your code submission

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

Locking Problems

- Administrative problems:



- Someone locks a given file and forgets about it

- Time is lost while waiting for someone to release a file

- Unneeded locking of the whole time
  - Different changes are not necessary in conflict
  - Example: Ali works on the beginning of the file and Samira works on the end

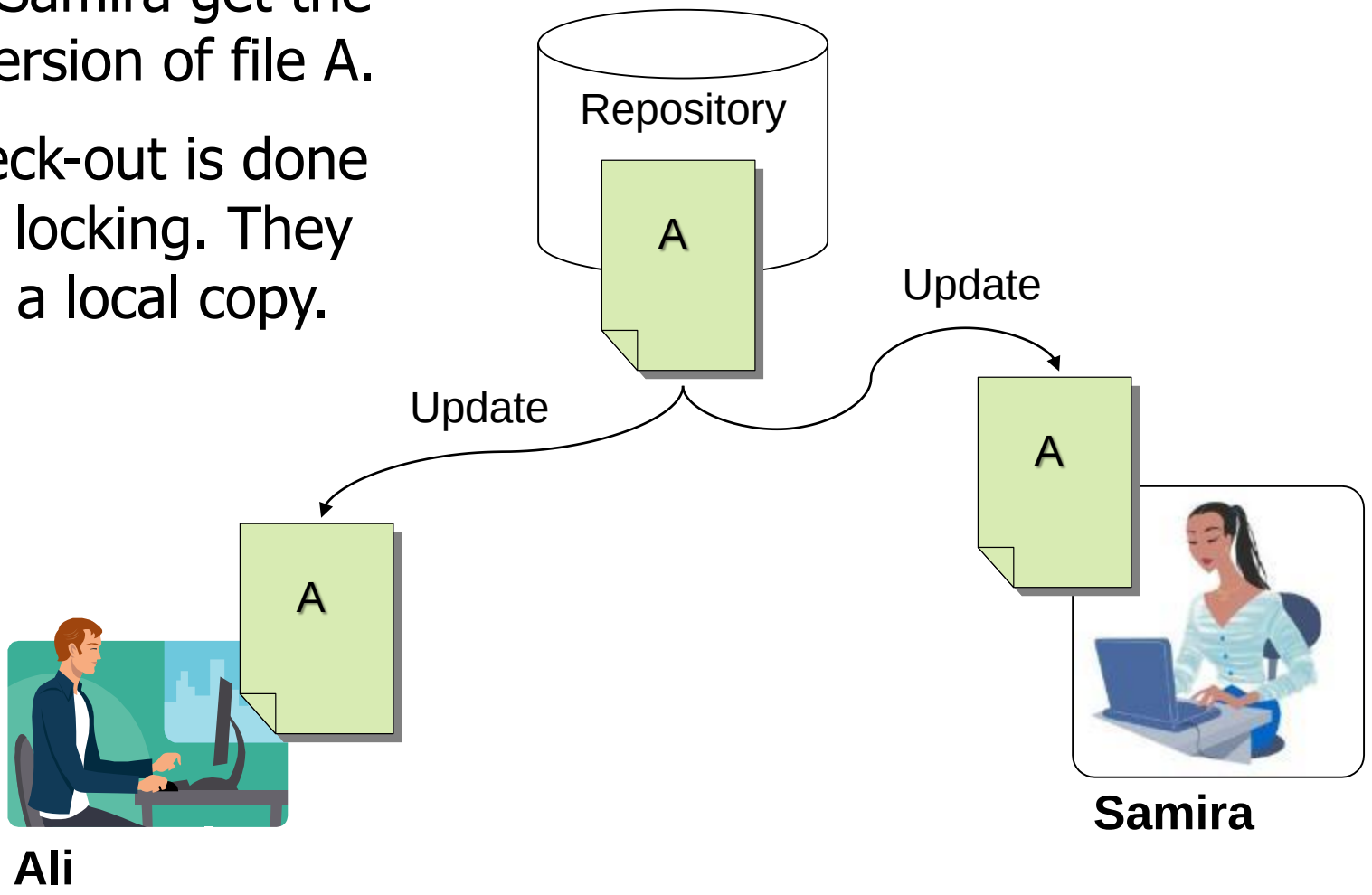
Merging Problems

- 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 Lock-Modify-Unlock Model (1)

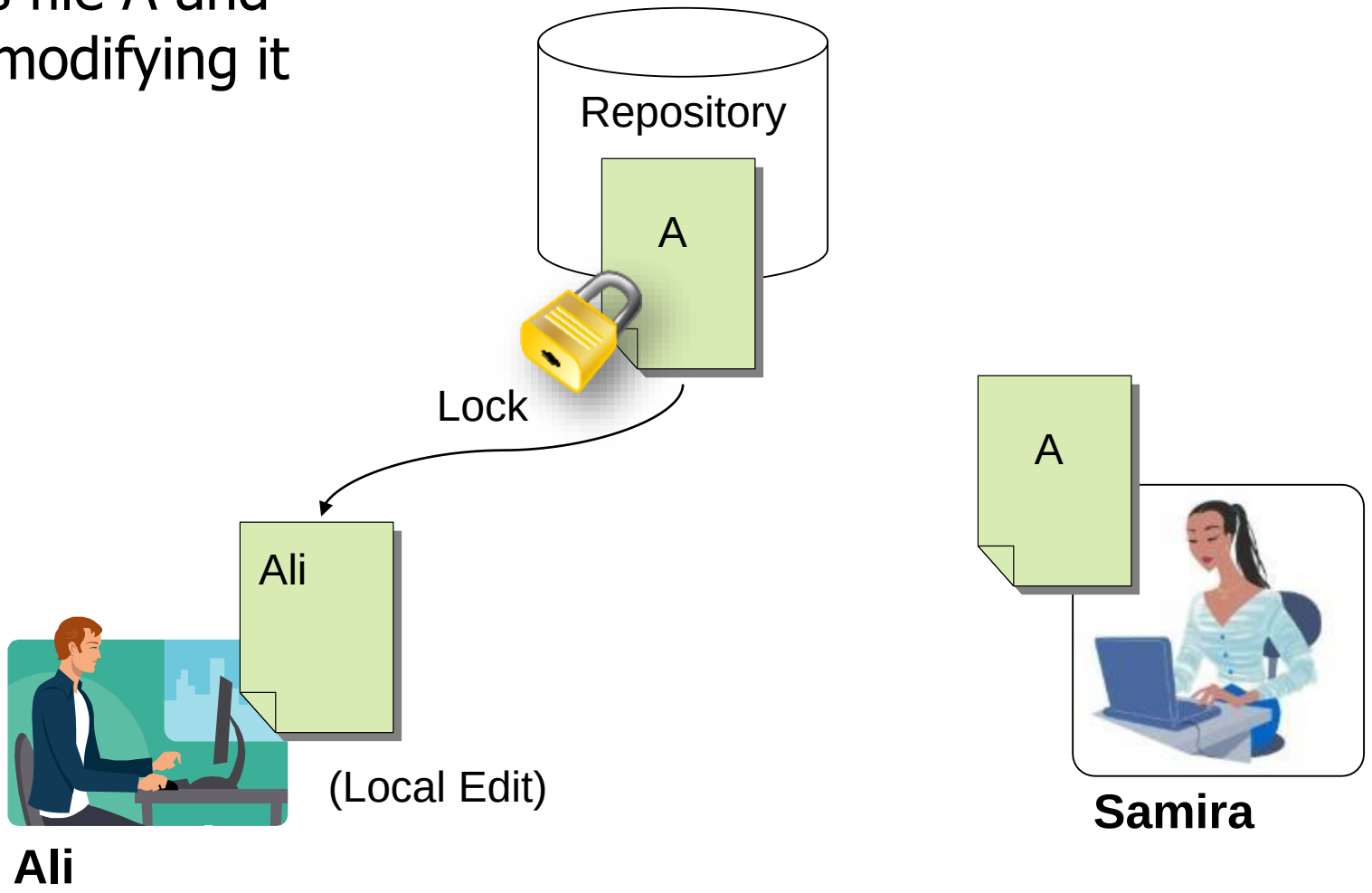
Ali and Samira get the latest version of file A.

The check-out is done without locking. They just get a local copy.



The Lock-Modify-Unlock Model (2)

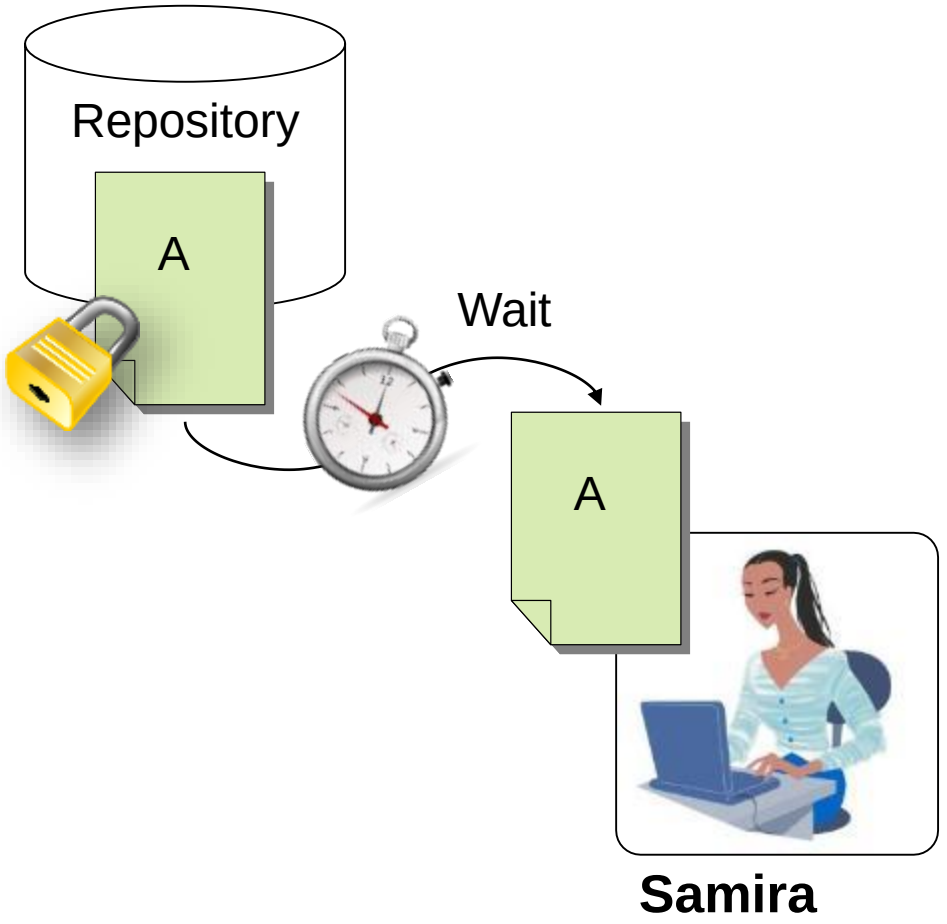
Ali locks file A and begins modifying it



The Lock-Modify-Unlock Model (3)

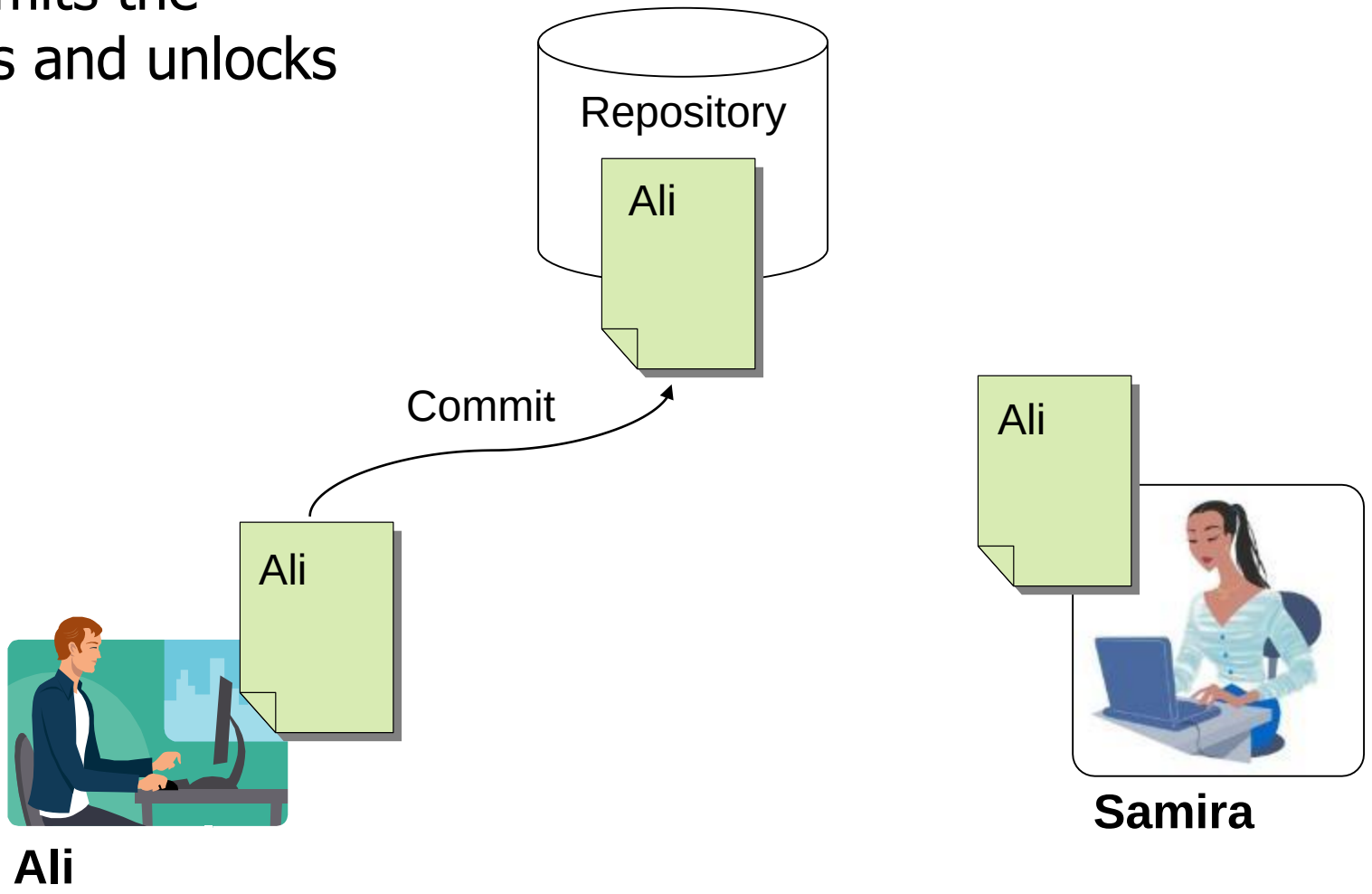
Samira tries to lock the file too, but she can't.

Samira waits for Ali to finish and unlock the file.



The Lock-Modify-Unlock Model (4)

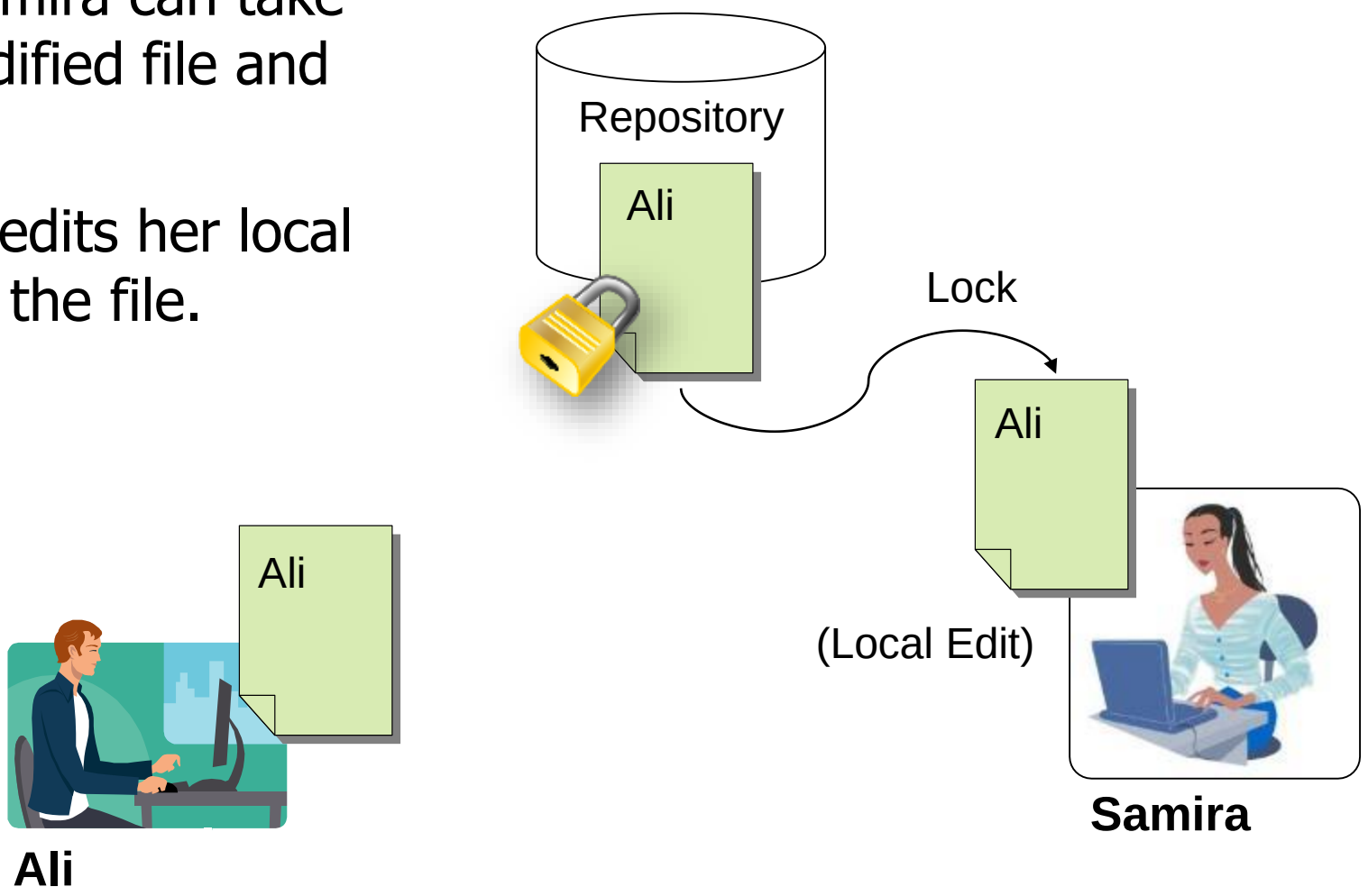
Ali commits the changes and unlocks the file.



The Lock-Modify-Unlock Model (5)

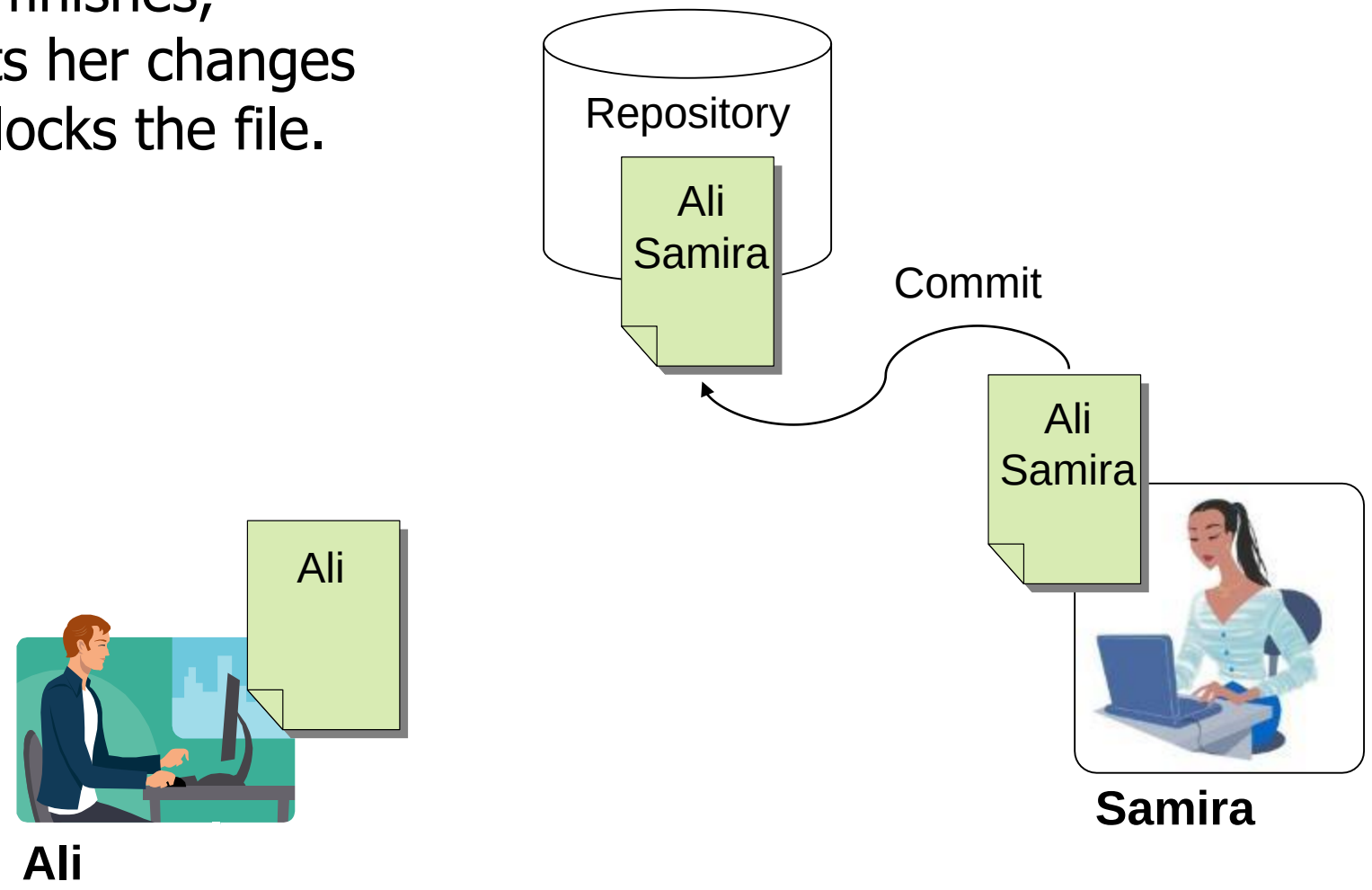
Now Samira can take the modified file and lock it.

Samira edits her local copy of the file.



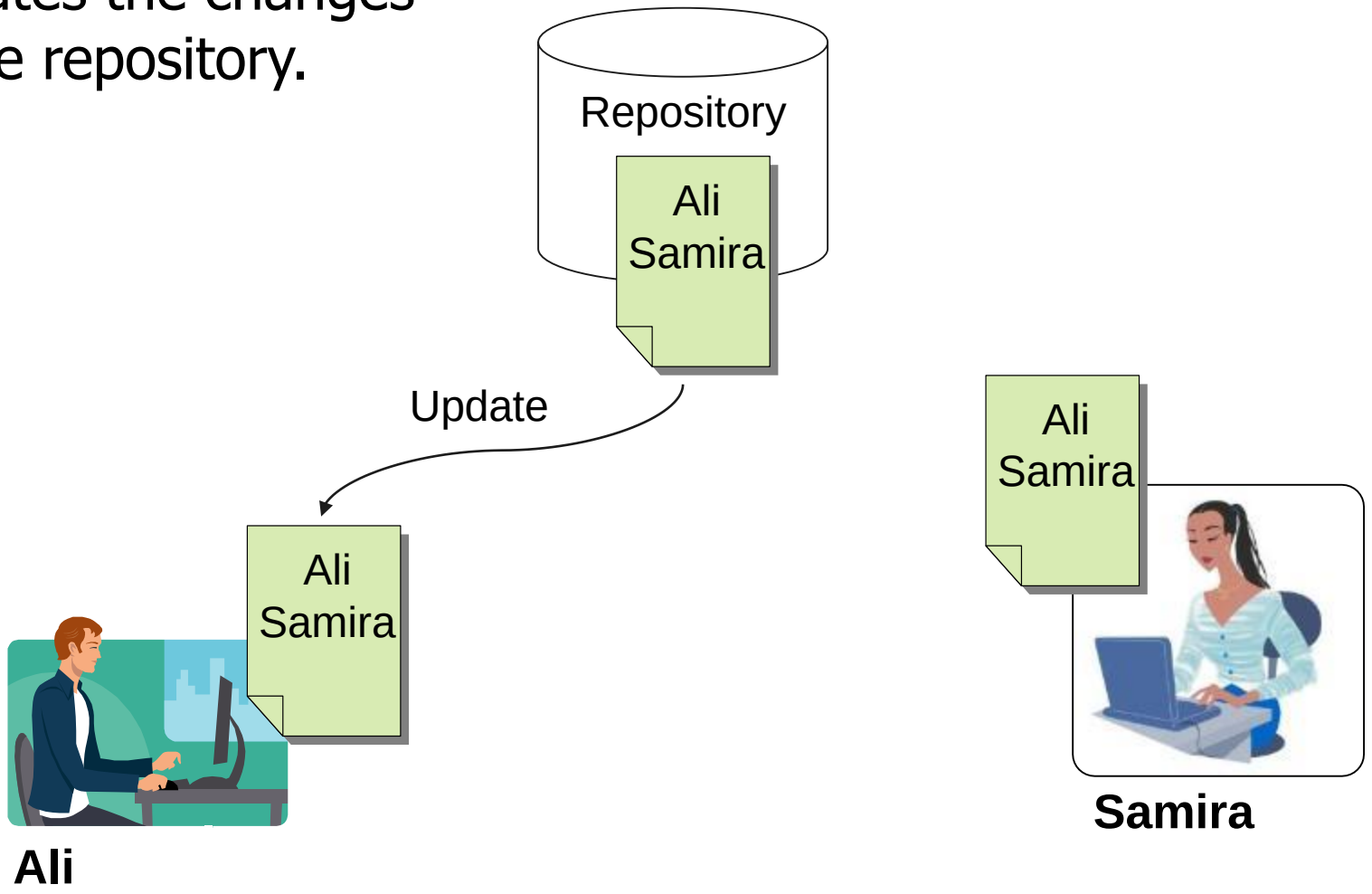
The Lock-Modify-Unlock Model (6)

Samira finishes, commits her changes and unlocks the file.



The Lock-Modify-Unlock Model (7)

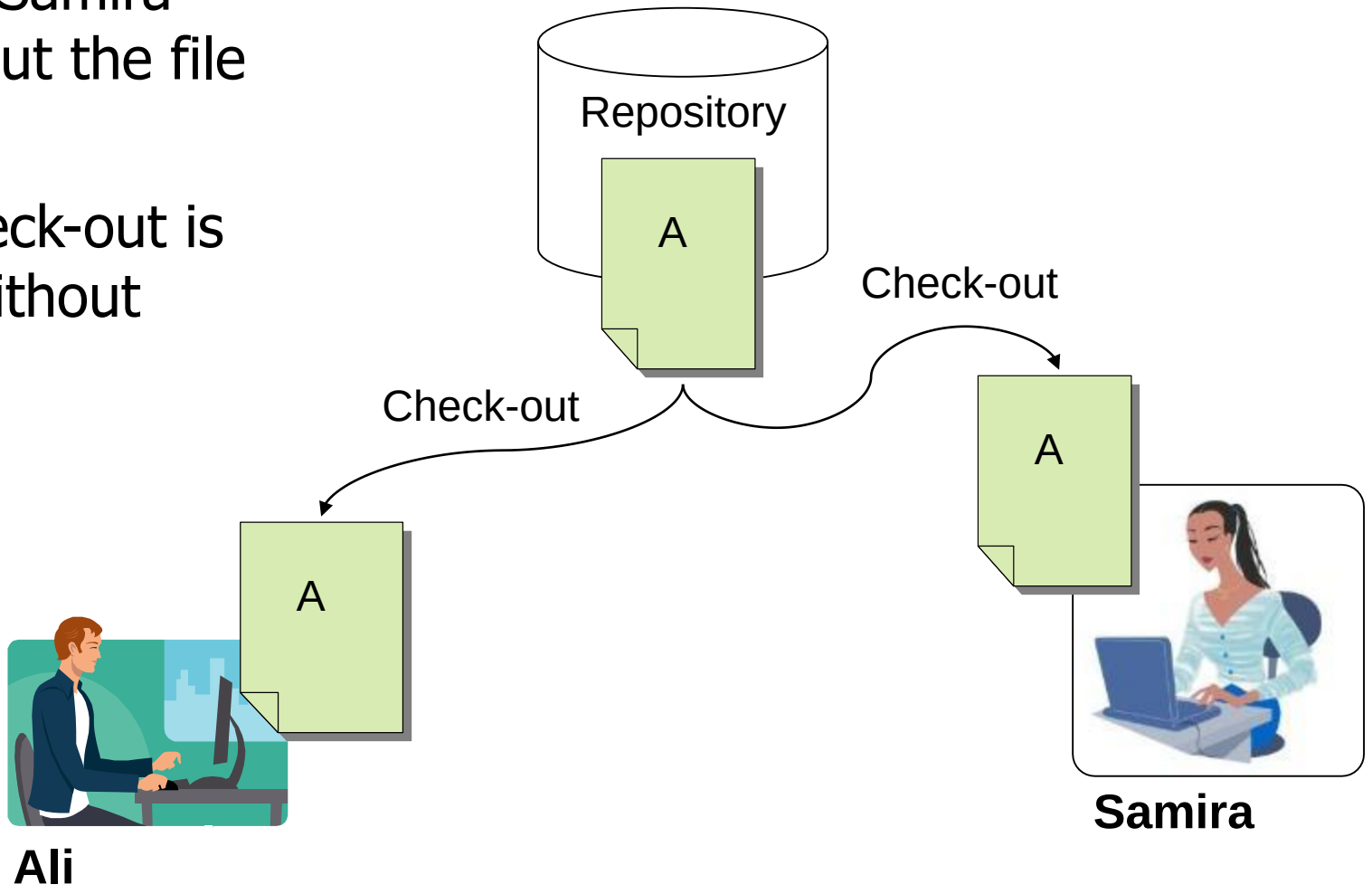
Ali updates the changes from the repository.



The Copy-Modify-Merge Model (1)

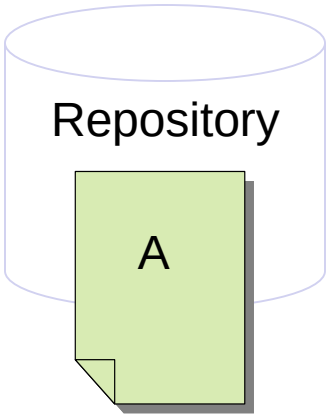
Ali and Samira check-out the file A.

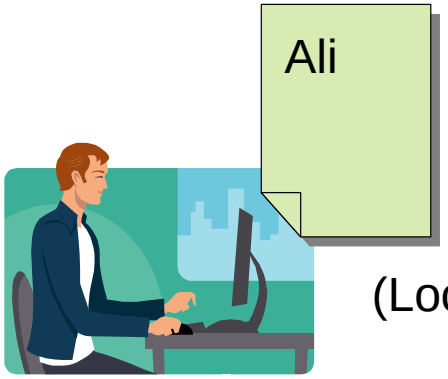
The check-out is done without locking.

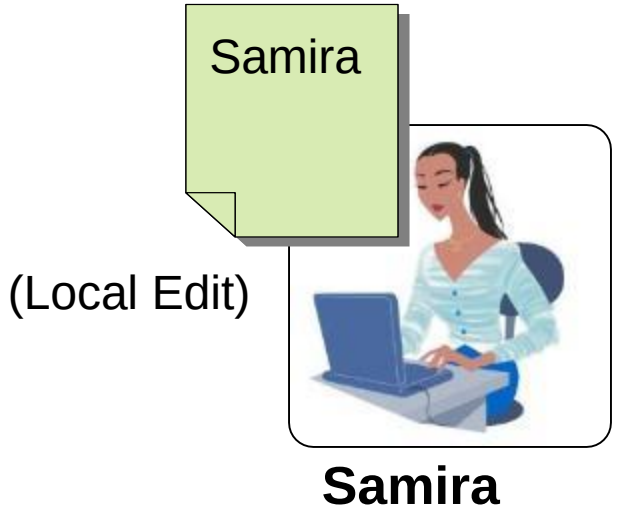


The Copy-Modify-Merge Model (2)

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

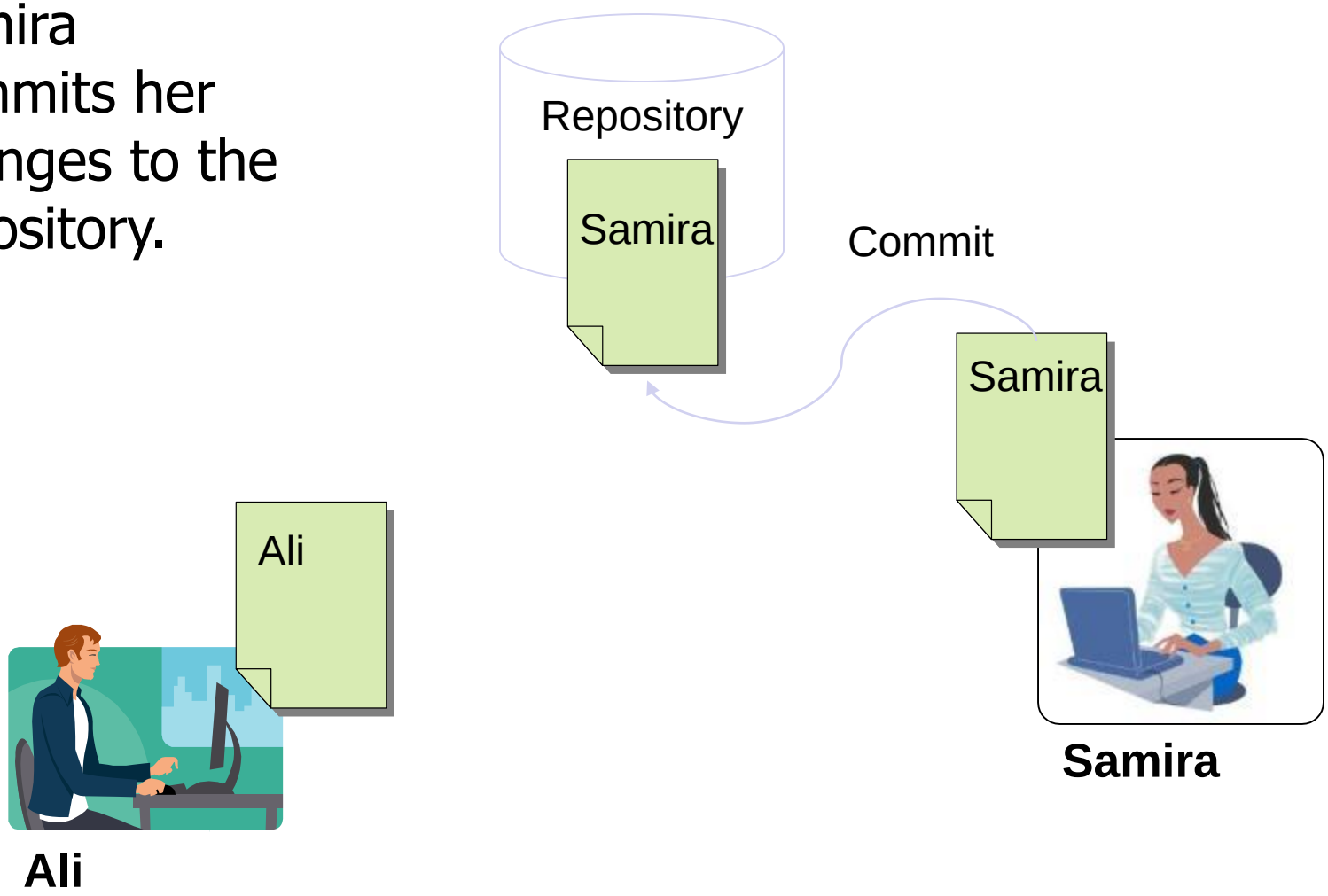






The Copy-Modify-Merge Model (3)

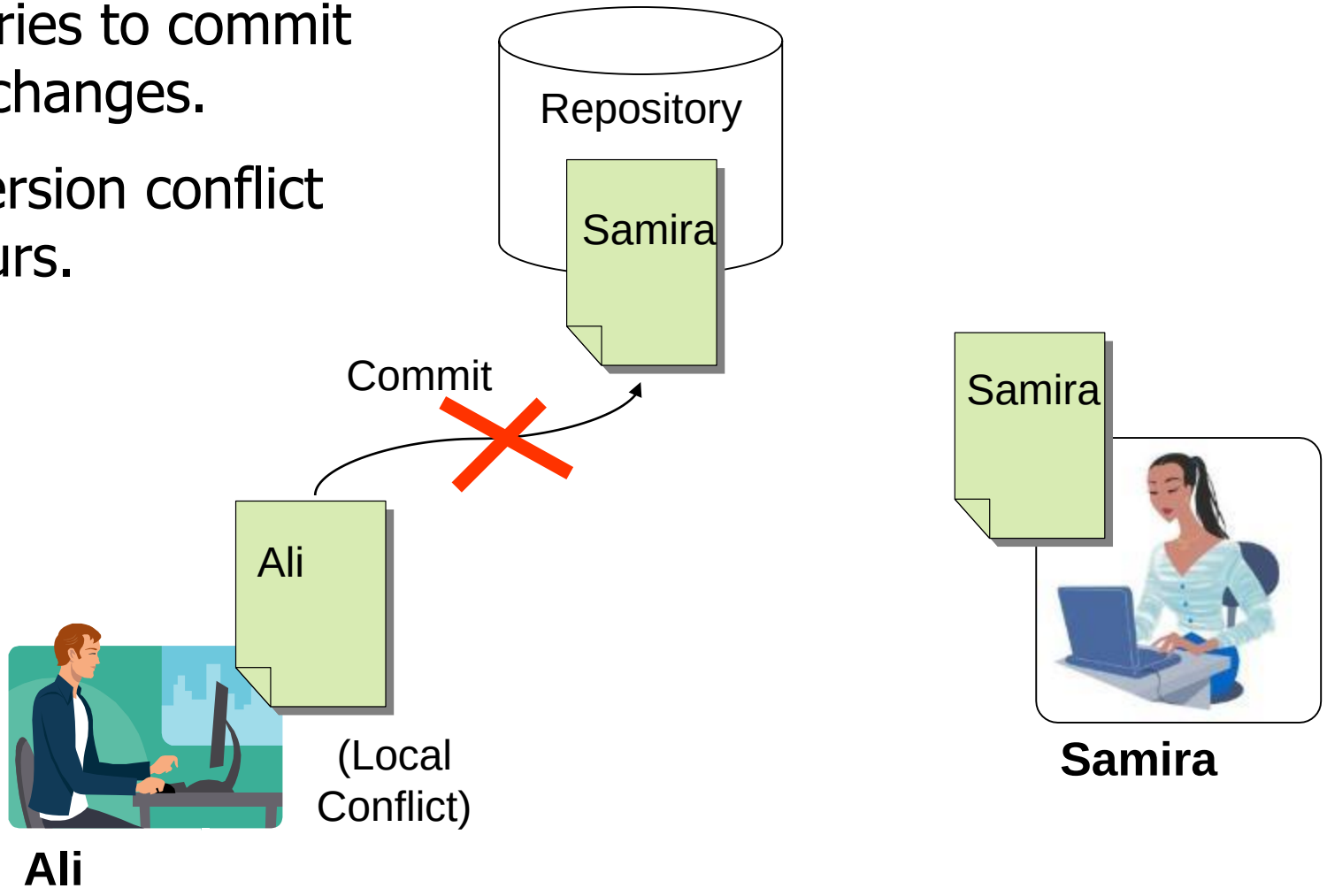
Samira commits her changes to the repository.



The Copy-Modify-Merge Model (4)

Ali tries to commit his changes.

A version conflict occurs.

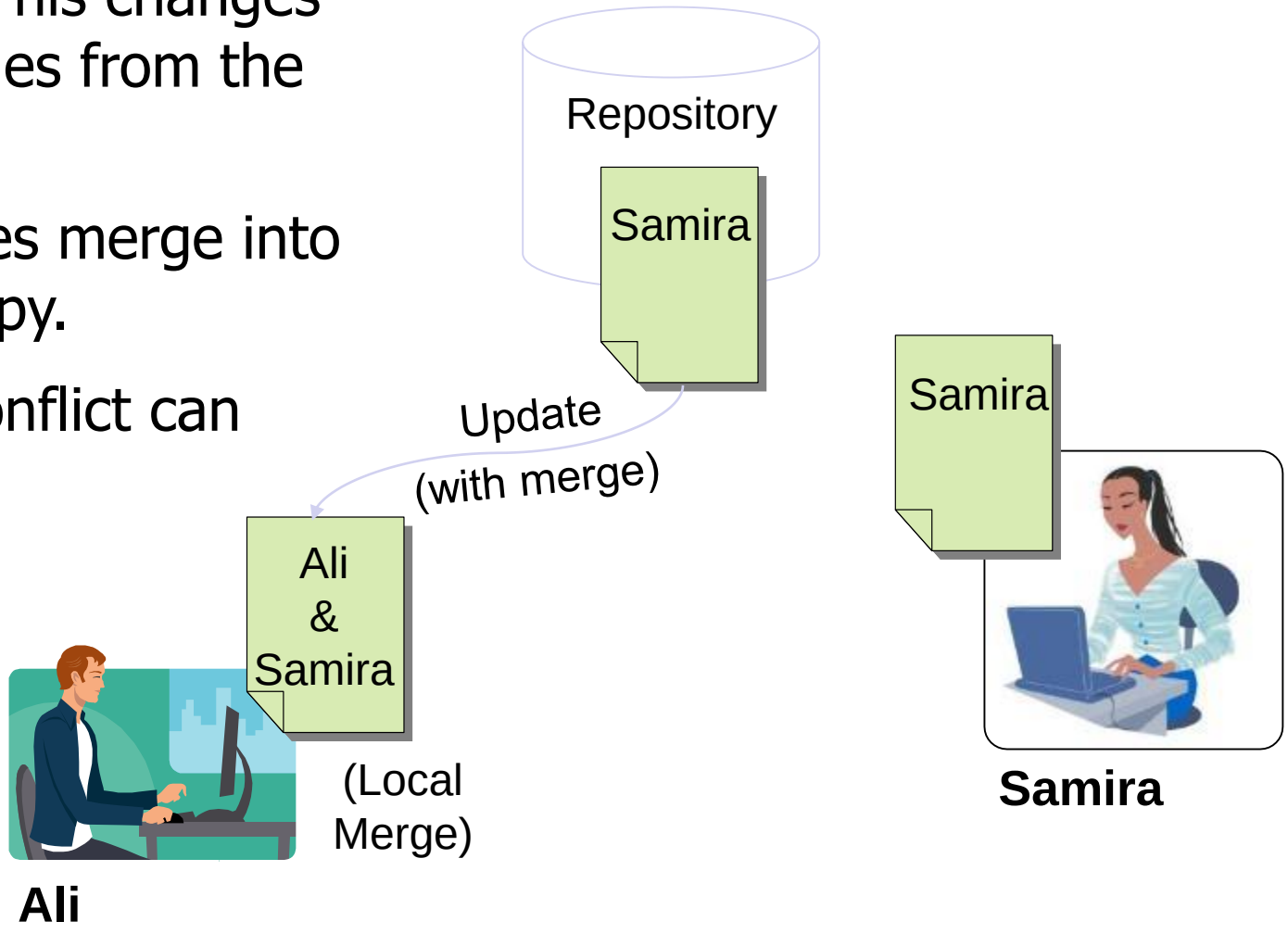


The Copy-Modify-Merge Model (5)

Ali updates his changes with the ones from the repository.

The changes merge into his local copy.

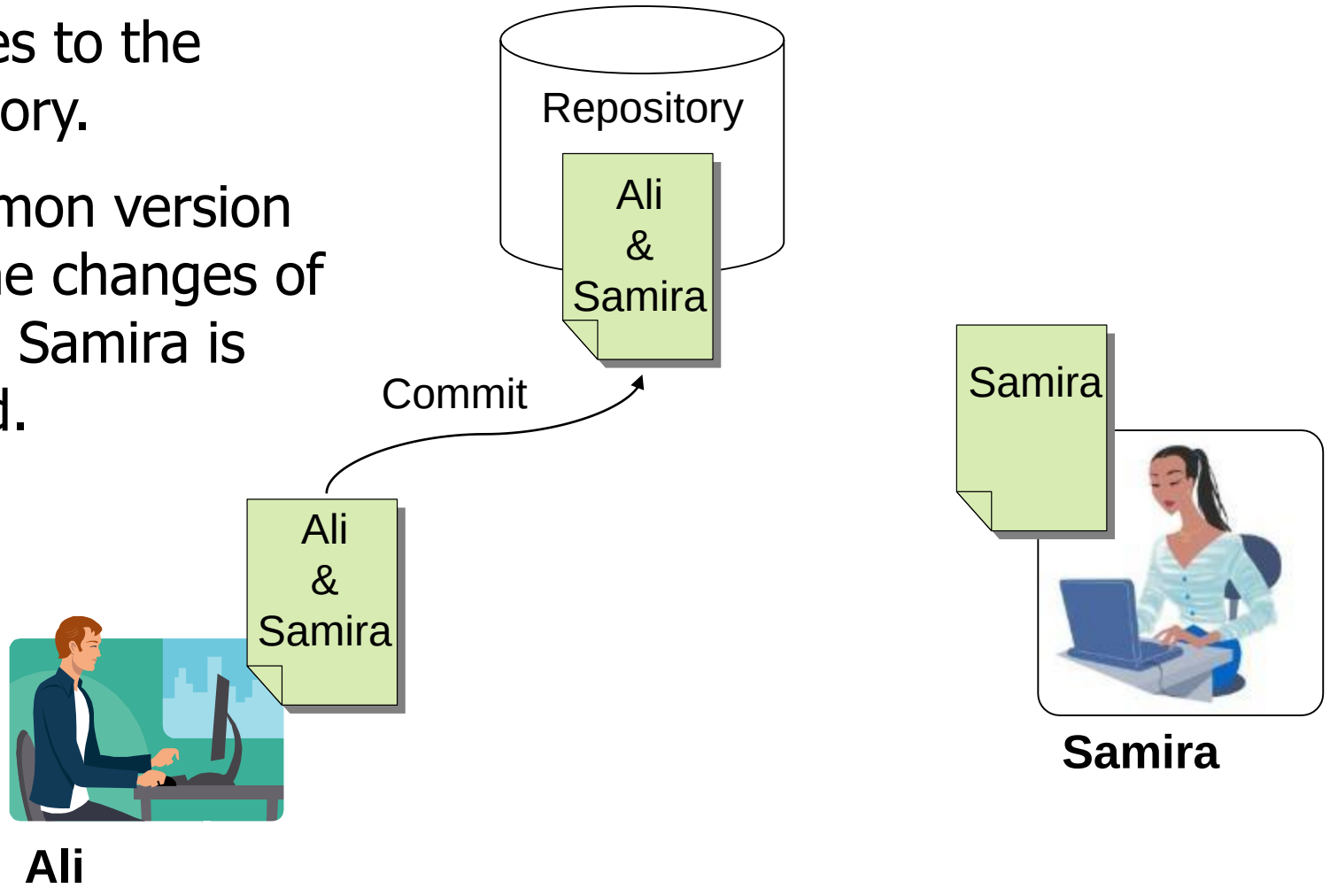
A merge conflict can occur.



The Copy-Modify-Merge Model (6)

Ali commits the changes to the repository.

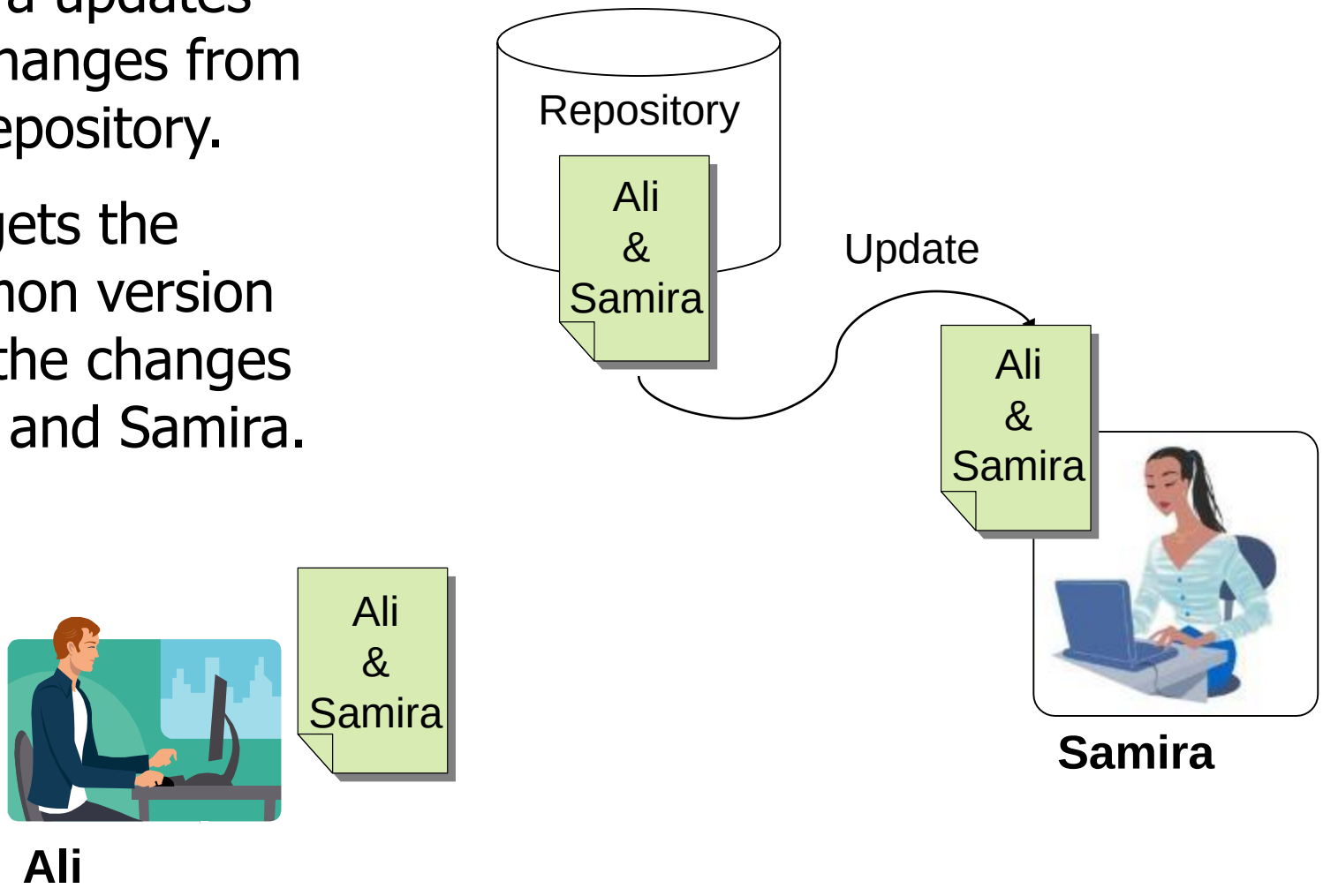
A common version with the changes of Ali and Samira is pushed.



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.

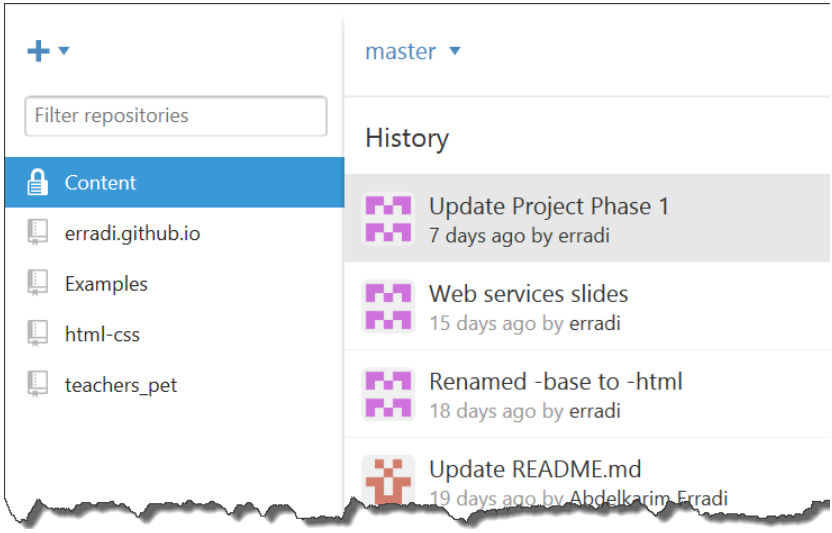


Project Hosting and Team Collaboration Sites

GitHub, SourceForge, CodePlex,
Visual Studio Online

Project Hosting Sites

- GitHub – https://github.com
  - The #1 project hosting site
  - Free for open-source projects
  - Paid plans for private projects
- GitHub provides Windows/Mac client
  - http://windows.github.com
  - https://mac.github.com/
  - Dramatically simplifies Git
  - For beginners only



Project Hosting Sites (2)

- SourceForge – http://www.sourceforge.net
  - Source control (SVN, Git, …), web hosting, tracker, wiki, blog, mailing lists, file release, statistics, etc.
  - Free, all projects are public and open source
- CodePlex – http://www.codeplex.com
  - Microsoft's open source projects site
  - Team Foundation Server (TFS) infrastructure
  - Source control (TFS), issue tracker, downloads, discussions, wiki, etc.
  - Free, all projects are public and open source

Project Hosting Sites (3)

- Visual Studio Online – https://www.visualstudio.com/products/what-is-visual-studio-online-vs

Source control (Team Foundation Version Control (TFVC), Git), issue tracker, etc.

  - Private / public projects, free and paid editions

- Bitbucket – http://bitbucket.org

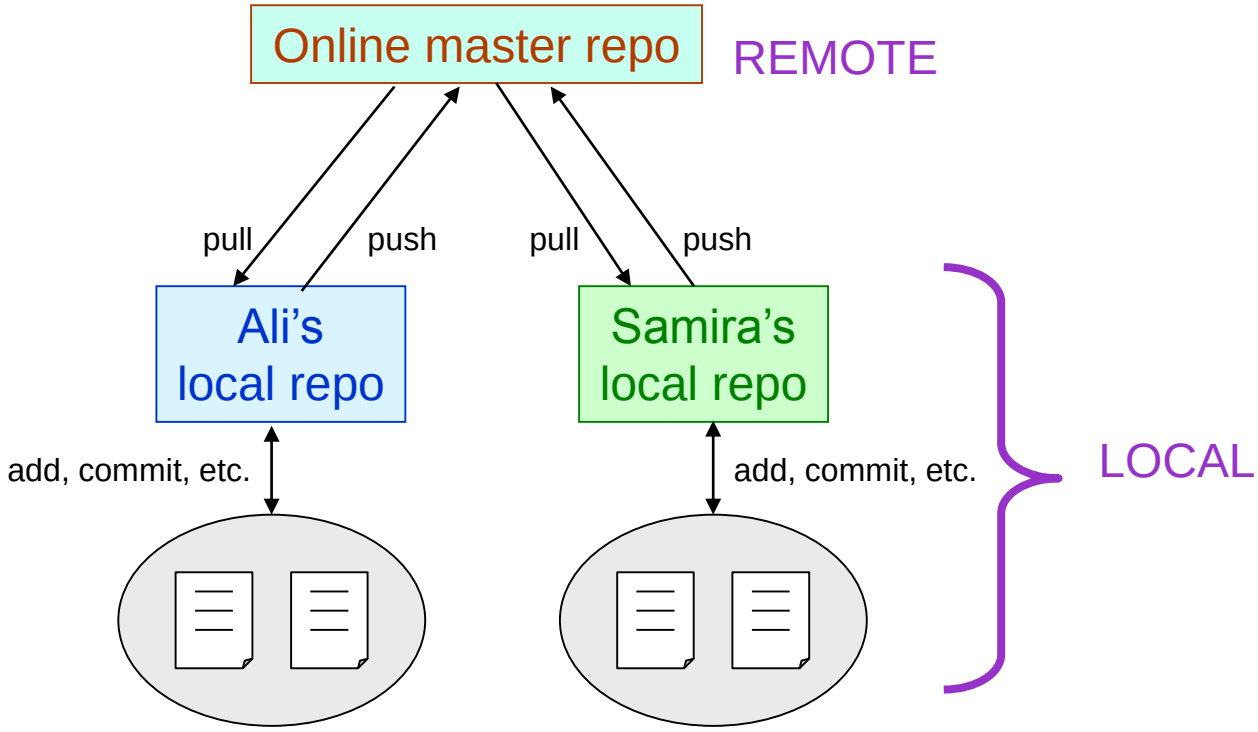
  - Source control (Mercurial, Git), issue tracker, wiki, management tools
  - Private projects, free and paid editions

Git and Github

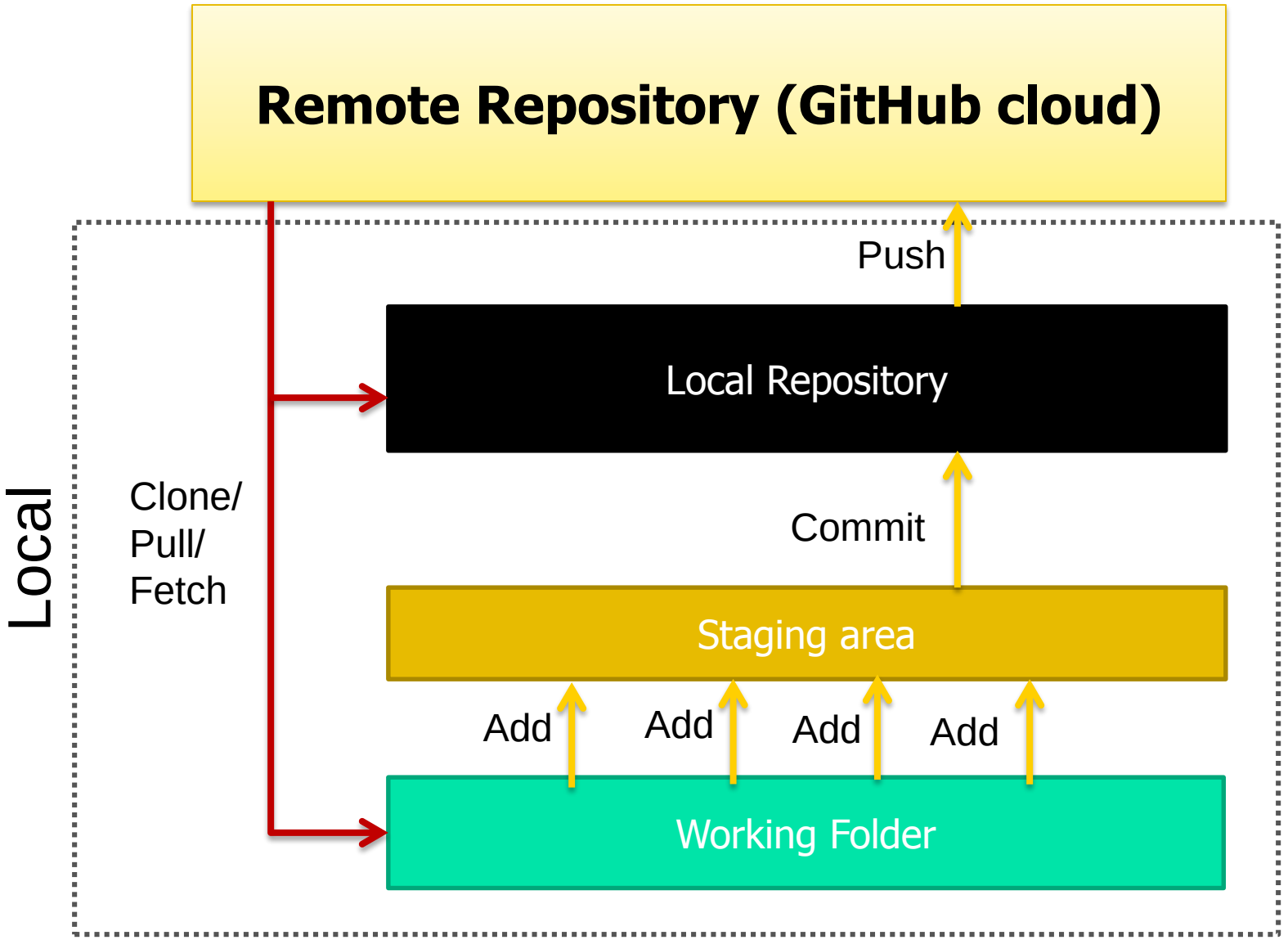
Github

- 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**

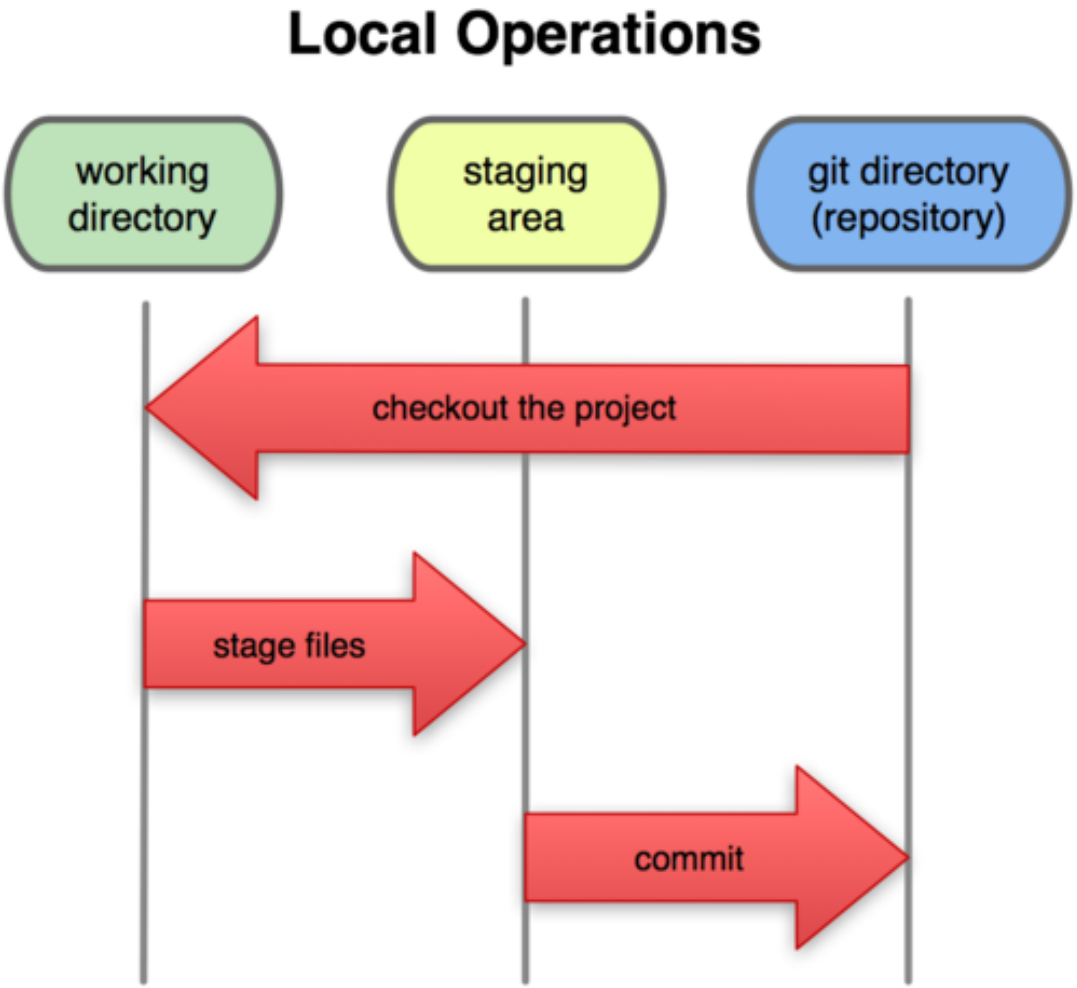
Local and Remote Repositories



Architecture & Terminology

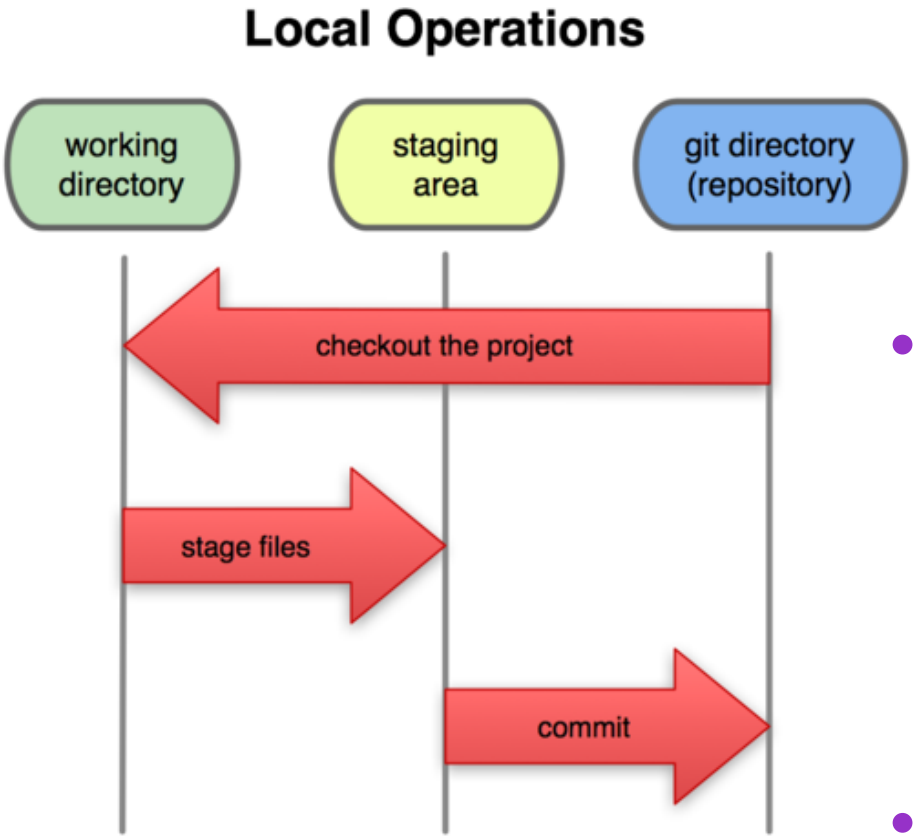


Git: Three File States



- A file can be in one of three states:
  - modified
  - staged
  - Committed
  - Untracked!
    - Newly added or removed files

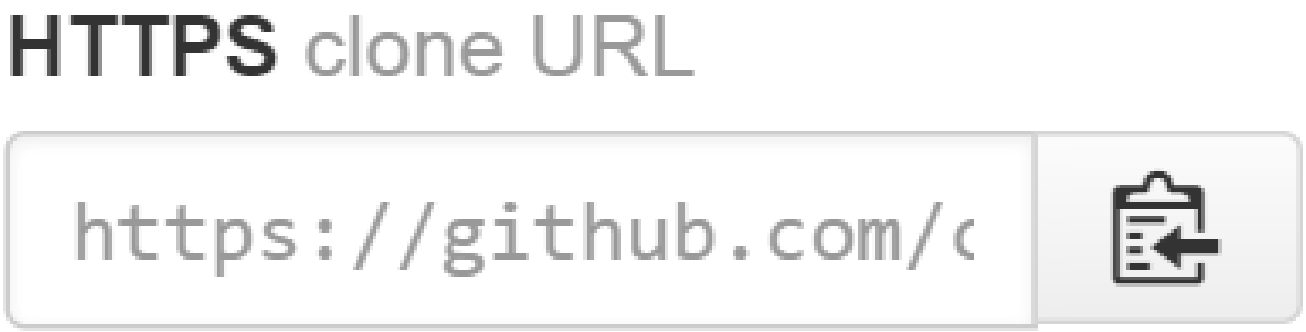
Git: Three File States *(cont'd)*



- The hidden Git directory `.git` is the local project repository
- The working directory is where you check out a version of the project for you to work on
- The staging area is a file that keeps track of what will go into your next commit

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.



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/cmps356s15/Examples.git
```

Git: Make Local Changes

- Get the status of files in your local repository:

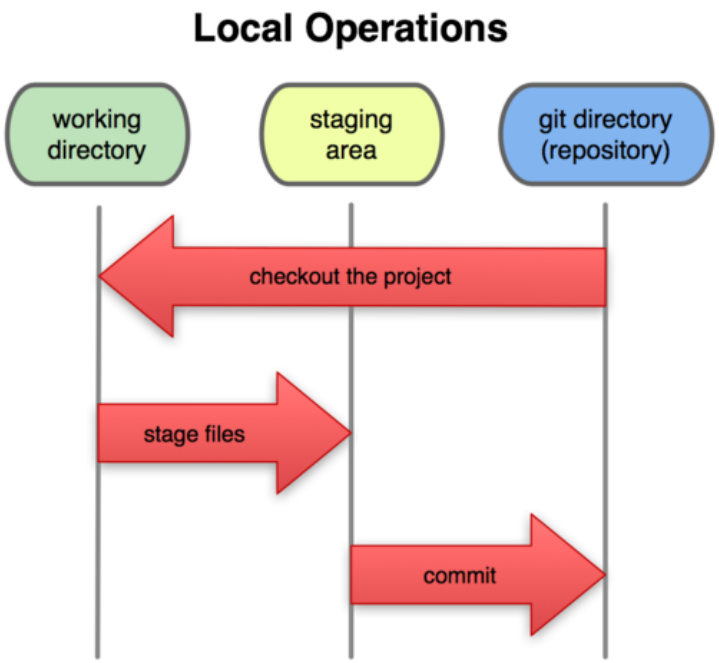
```
git status
```

- After you've modified files (or created new files) on your local machine, first add them to the local staging area:

```
git add myfile1 myfile2
```

- Commit your staged files to the local repository:

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



Git: Push to the Master Repository

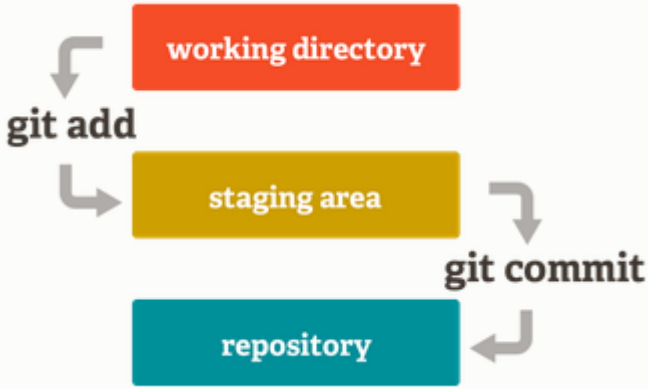
- From the directory of the local repository, push your local repository contents up to the master repository at GitHub.

```
git push
```

- 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.

```
git pull
```

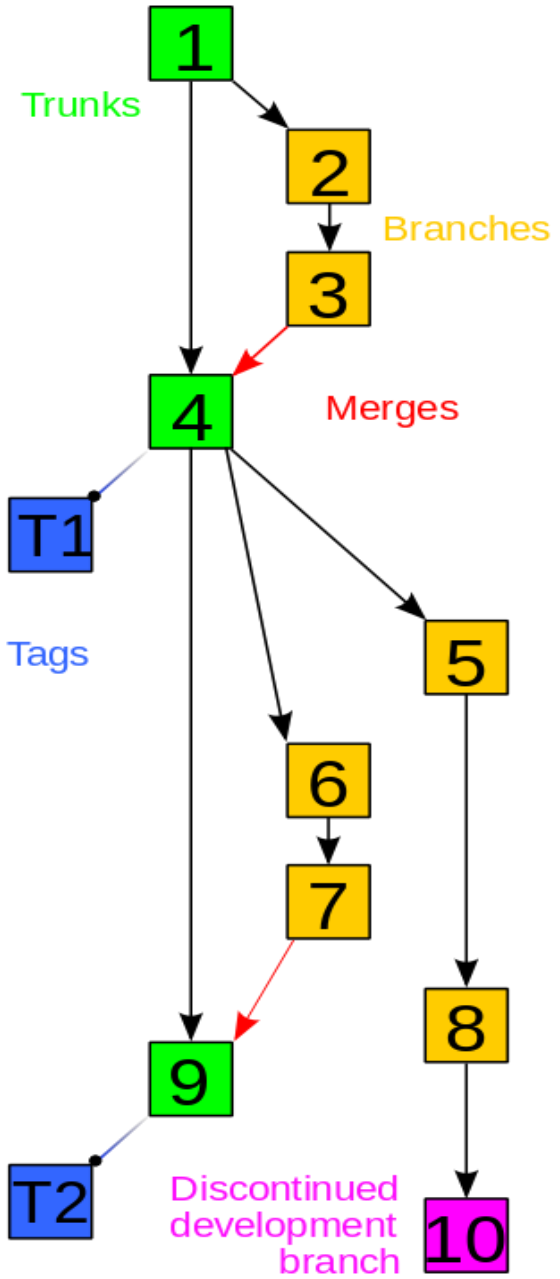
Git Basic Commands Summary



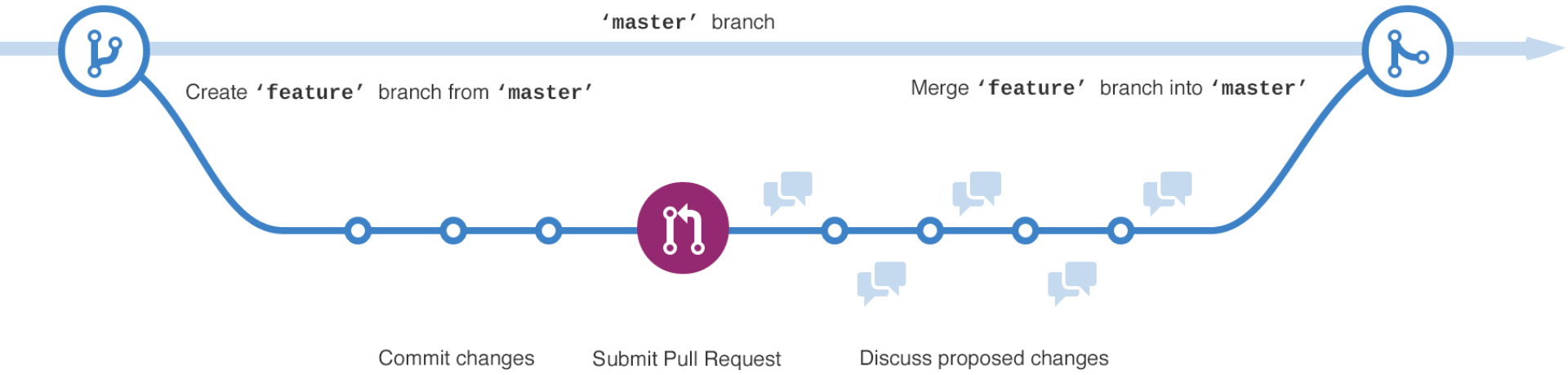

```
git init    //initializes git

git add filename  //adds file name

git diff    //prints difference made in files

git commit -m "Message here "  //saved!!

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
```

History Tree

- Trunk is in green
- 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



GitHub Example



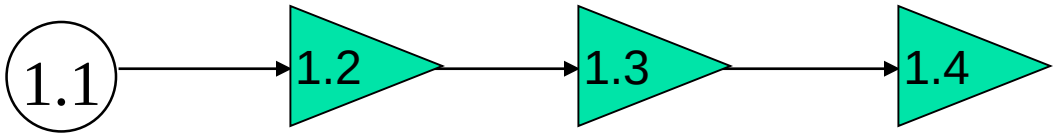
- Developers use branches for keeping bug fixes and feature work **separate** from the ***master*** (production) branch. When a feature or fix is ready, the branch is merged into master.
- Before merge you may make a **pull request**, to start a discussion about commits (code review) and get feedback

Branches

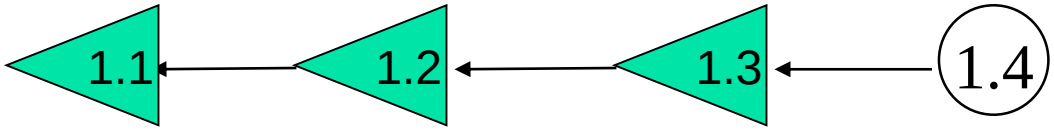
- **Isolating** new development from finished work
- New development (new features, non-emergency bug fixes) are built in **feature branches.** They are only **merged** back into master branch when ready for release

Techniques for Storing Versions

- The set of differences between two versions is called a **delta**
- Forward delta

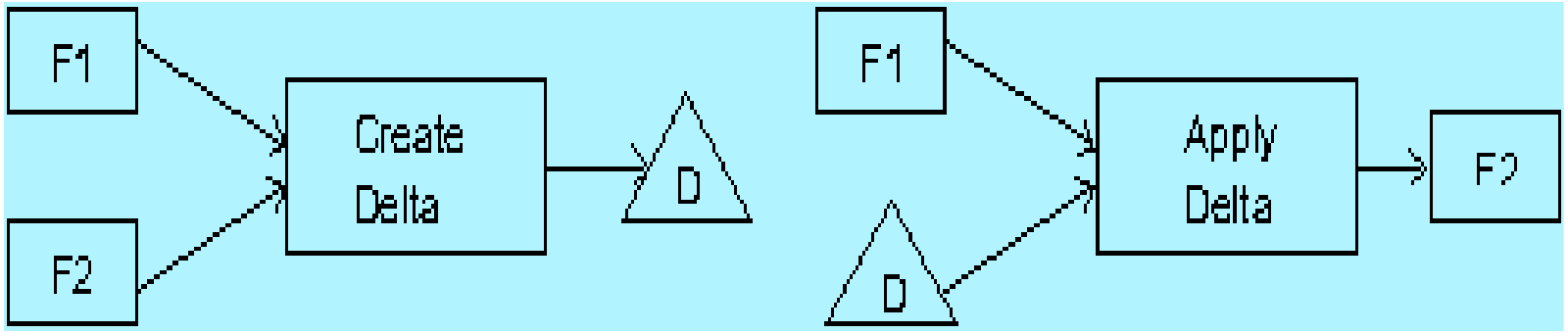


- Reverse delta

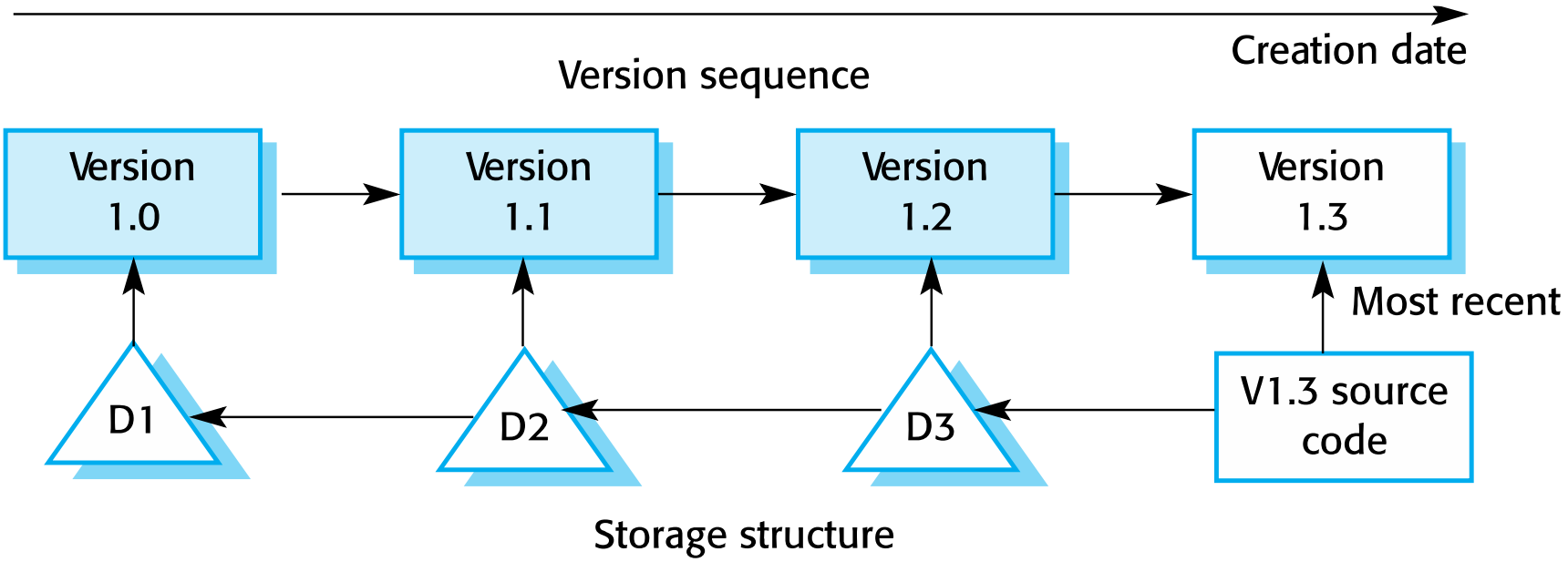


Constructing Delta

- Differential file or delta is formed from two original files (versions) by computing the difference between them.
- Let us assume that *F1* and *F2* are files and *D* is the delta to be generated.
- *D* must contain all the information to reconstruct *F2* from *F1*
- If *F1* is older version than *F2* (in respect to time of creation) *D* is called a *forward delta*.
- In the opposite case, when *F1* is newer version than *F2*, D is called a *reverse delta*.



Storage Management using Deltas



Forward Delta

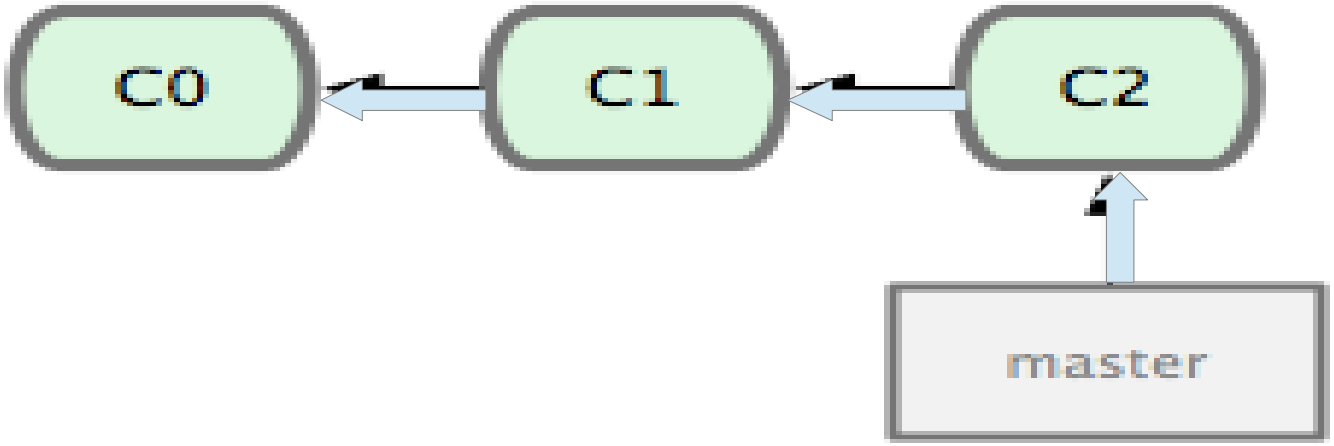
- The simplest delta technique is known as the forward delta
- Differences between two files are created in such a way that delta is applied to transform the old version into the new one.
- The forward deltas are calculated between the first and second versions, the second and third versions, etc.
- Only the first version is stored intact
- To restore n^{th} version the first, second,… $(n-1)^{th}$ deltas are applied to the first version.

Reverse Delta

- In reverse delta, the most recent version is accessed much more frequently than earlier versions.
- Deltas are arranged in such a way that the most recent revision on the trunk is stored intact.
- All other revisions on the trunk are stored as reverse deltas.
- Extraction of the latest version together with adding a new version to the trunk is fast

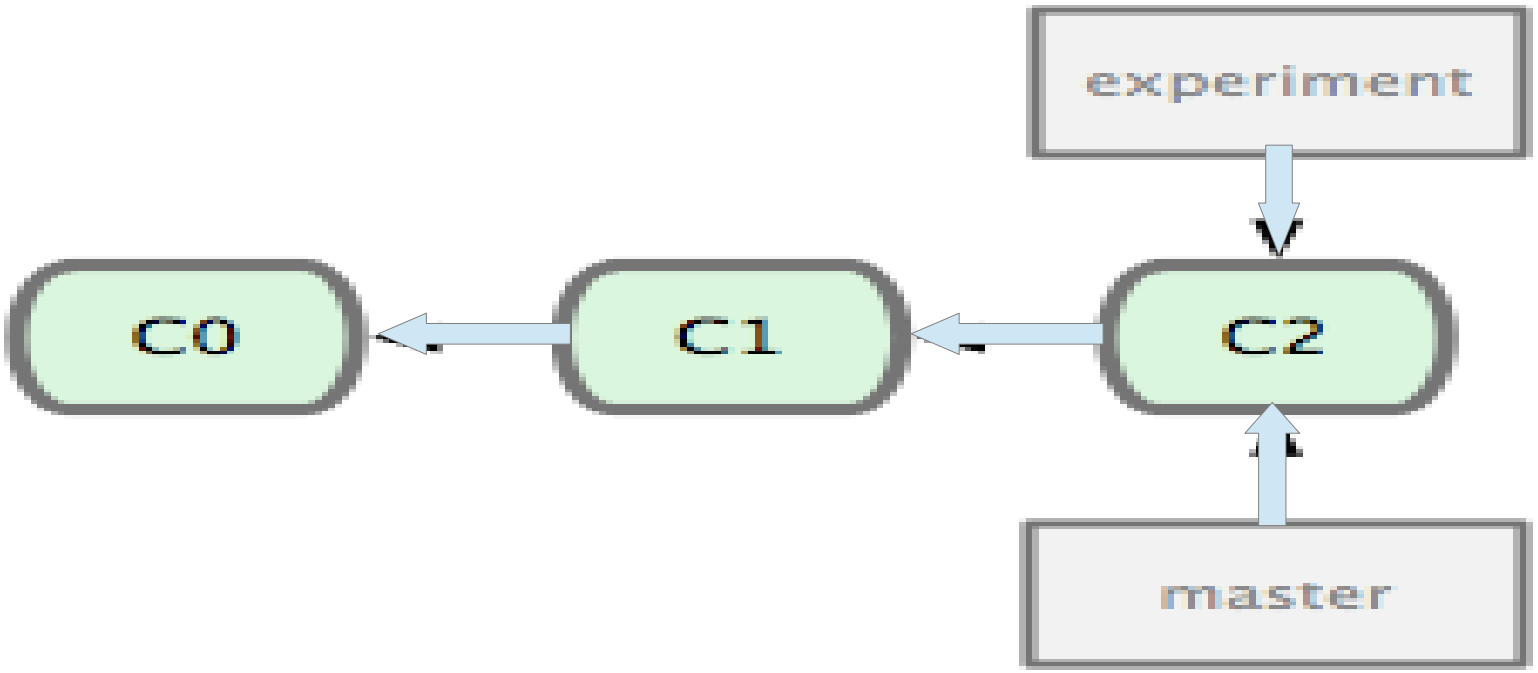
Git Branching & Merging

```
git  branch   //Shows all branches of current repository
```



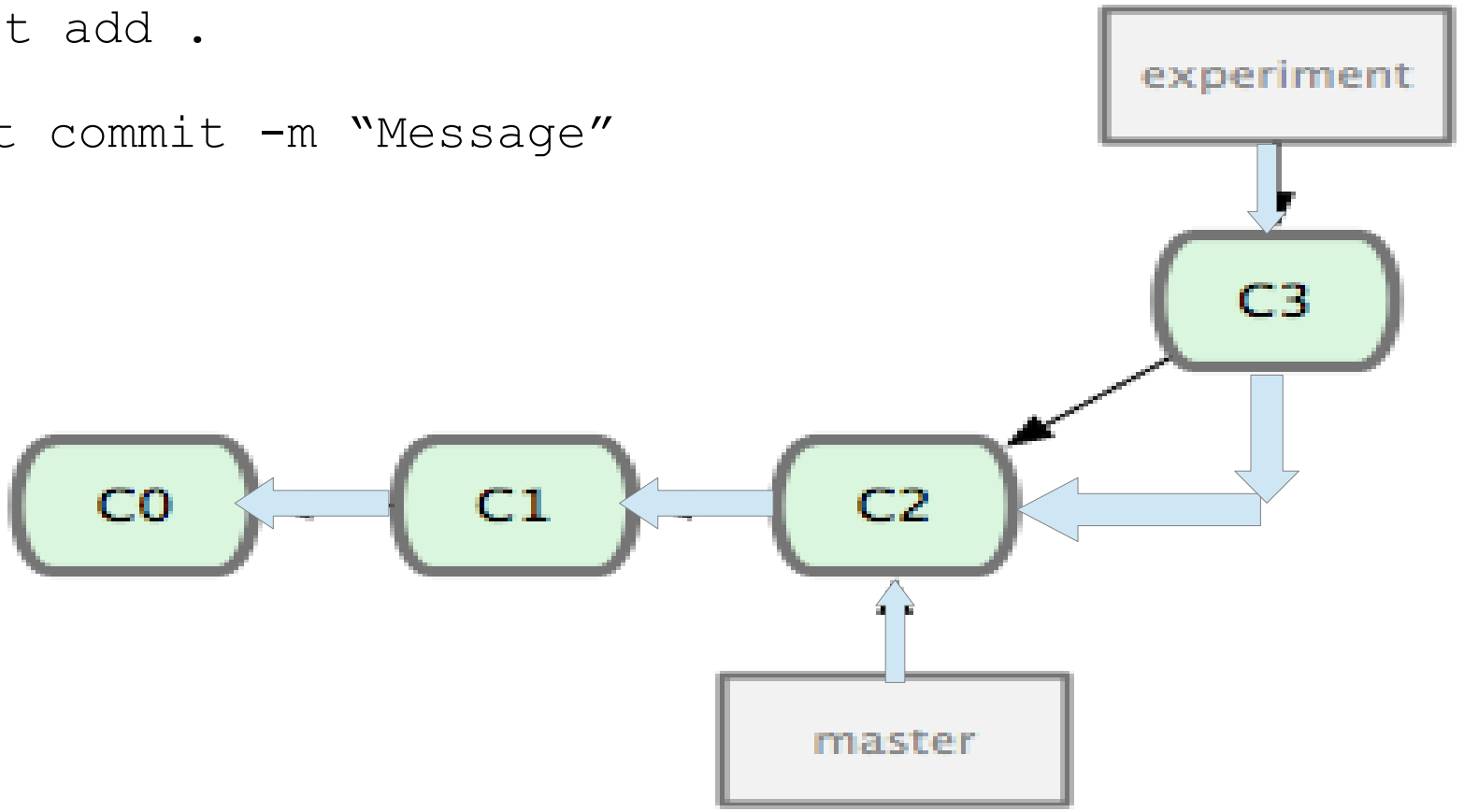
Git creating branch

```
git branch branch_name
```



Commit in Branch

```
git checkout branch_name    //Goes to the branch

 git add .

git commit -m "Message"
```



Merging

```
git merge master branch_name
//Used to merge your work with master
```

```
git checkout -b branch_name
//Creates new branch and goes
to that branch
```

```
git branch -d branch_name
 //Deletes the given branch
```