GIT

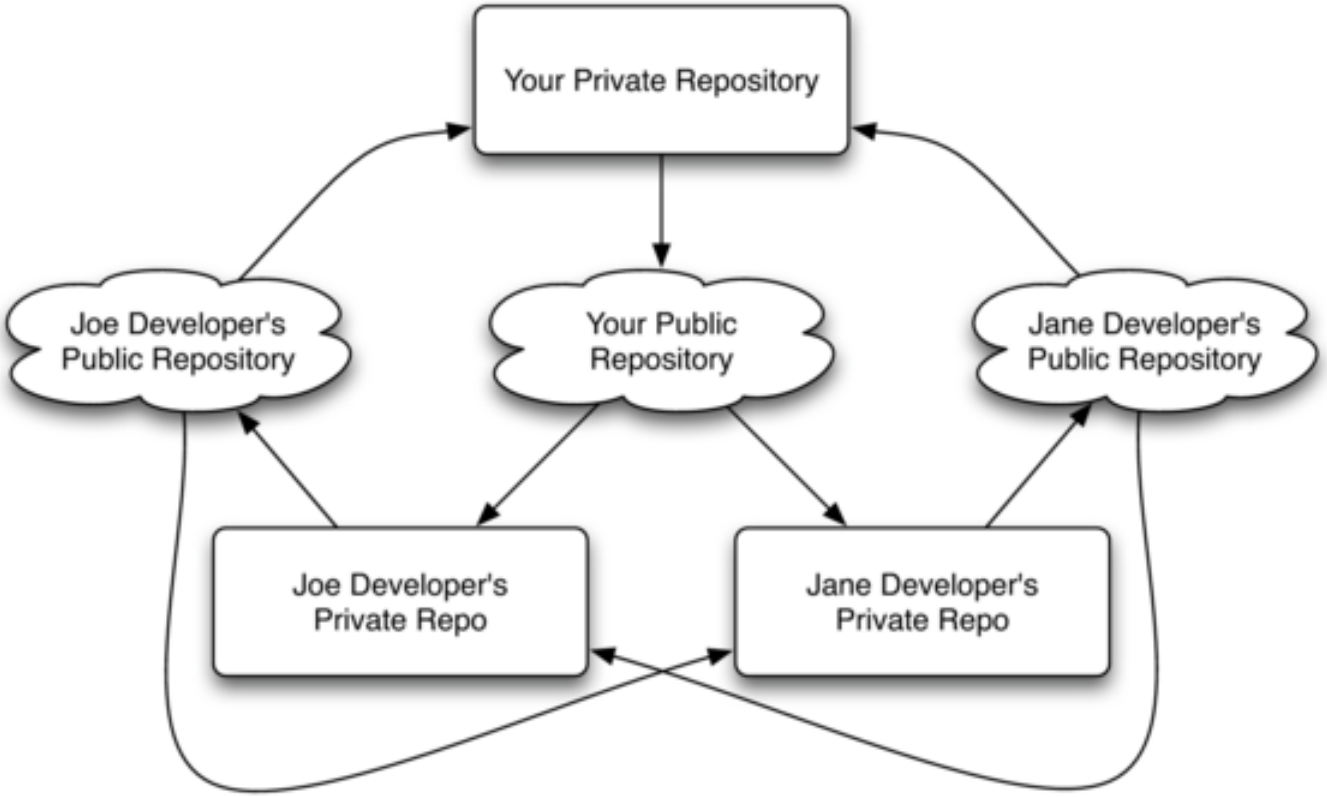
CS 490MT/5555, Spring 2016, Yongjie Zheng

GIT

- Overview
  - GIT Basics
  - Highlights: snapshot, the three states
- Working with the Private (Local) Repository
  - Creating a repository and making changes to it
- Working with the Public (Remote) Repository
  - Accessing protocols
  - Fetching, pulling, and pushing
- GIT Branching
- GIT Working Flows

Recall what we have learned

- Decentralized Repository Model



GIT Basics

- History
  - Developed by Linus Tornalds, the creator of Linux, in 2005.
  - Keywords: performance, distributed, trust
- Installation
  - http://git-scm.com
- Setting up

  git config --global user.name "Jack Smith" (specify your name)

  git config --global user.email jack@hotmail.com (specify email)

  git config --global core.editor emacs (specify the default editor)

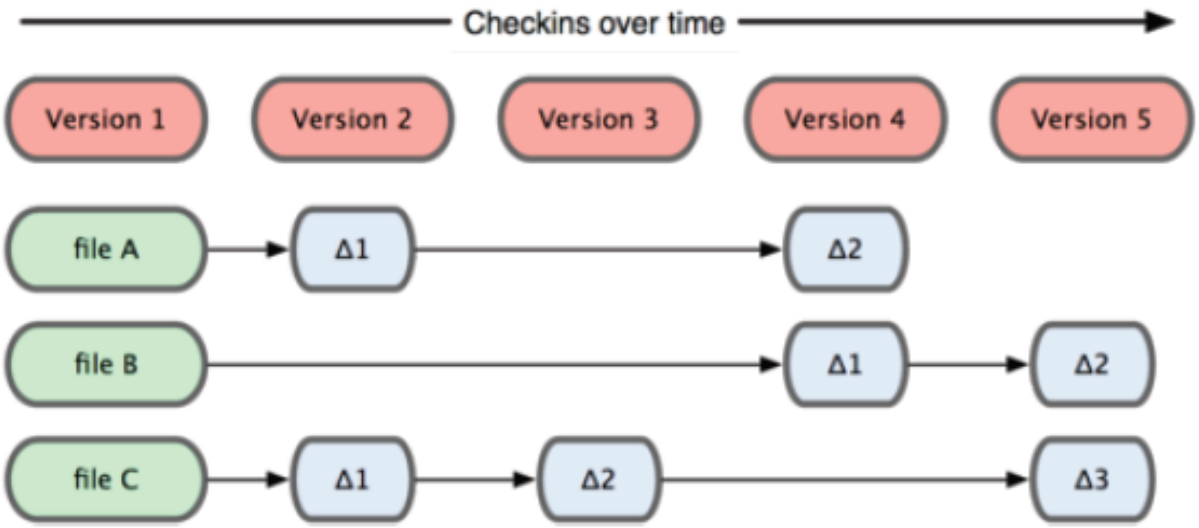
  git help <verb> (get help)

GIT Highlights

- Snapshots, Not Differences
  - Most version control systems store a set of files and the changes made to each file over time.
  - GIT saves the state of your project: every time you commit, GIT records (i.e. takes a picture of) all the files and stores a reference to that snapshot.
  - If files have not changed, GIT stores just a link to the previous identical file it already stored.

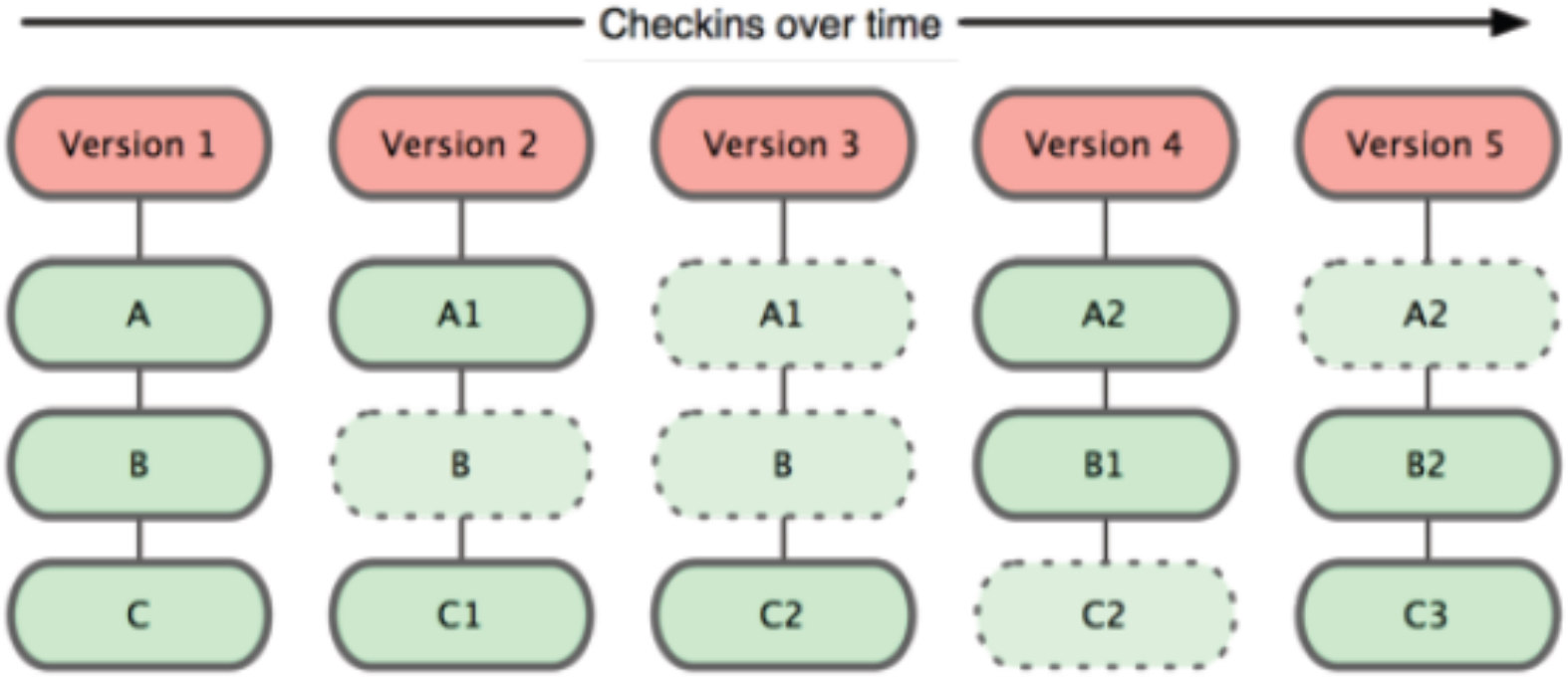
GIT Highlights

Figure 1.4: Other systems tend to store data as changes to a base version of each file.



GIT Highlights

Figure 1.5: Git stores data as snapshots of the project over time.

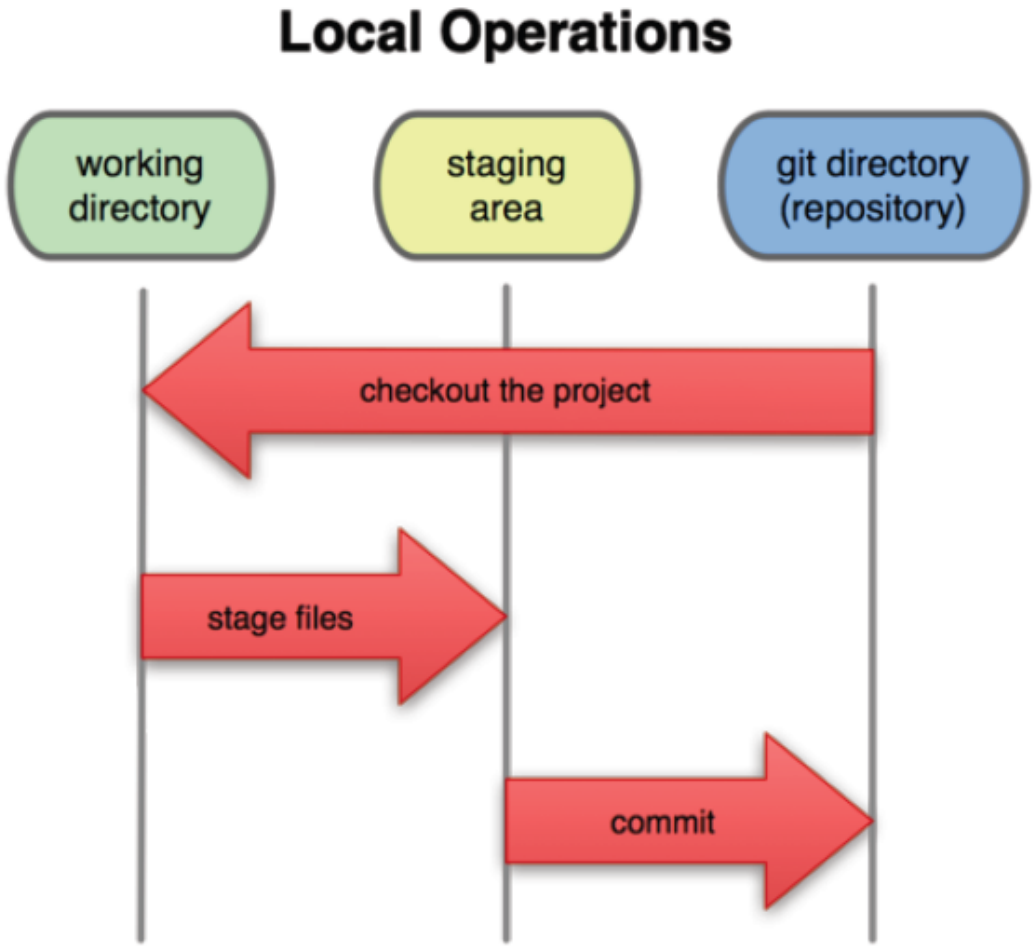


GIT Highlights

- The Three States
  - **Committed**: the data is safely stored in your local database.
  - **Modified**: you have changed the file but have not committed it to your database yet.
  - **Staged**: you have marked a modified file in its current version to go into your next commit snapshot.
- The GIT directory: the object database for your project.
- The working directory: a single checkout of one version of the project that is ready to be modified.
- The staging area: a simple file that stores information about what will go into your next commit.

GIT Highlights

Figure 1.6: Working directory, staging area, and git directory



Working with the Private Repository

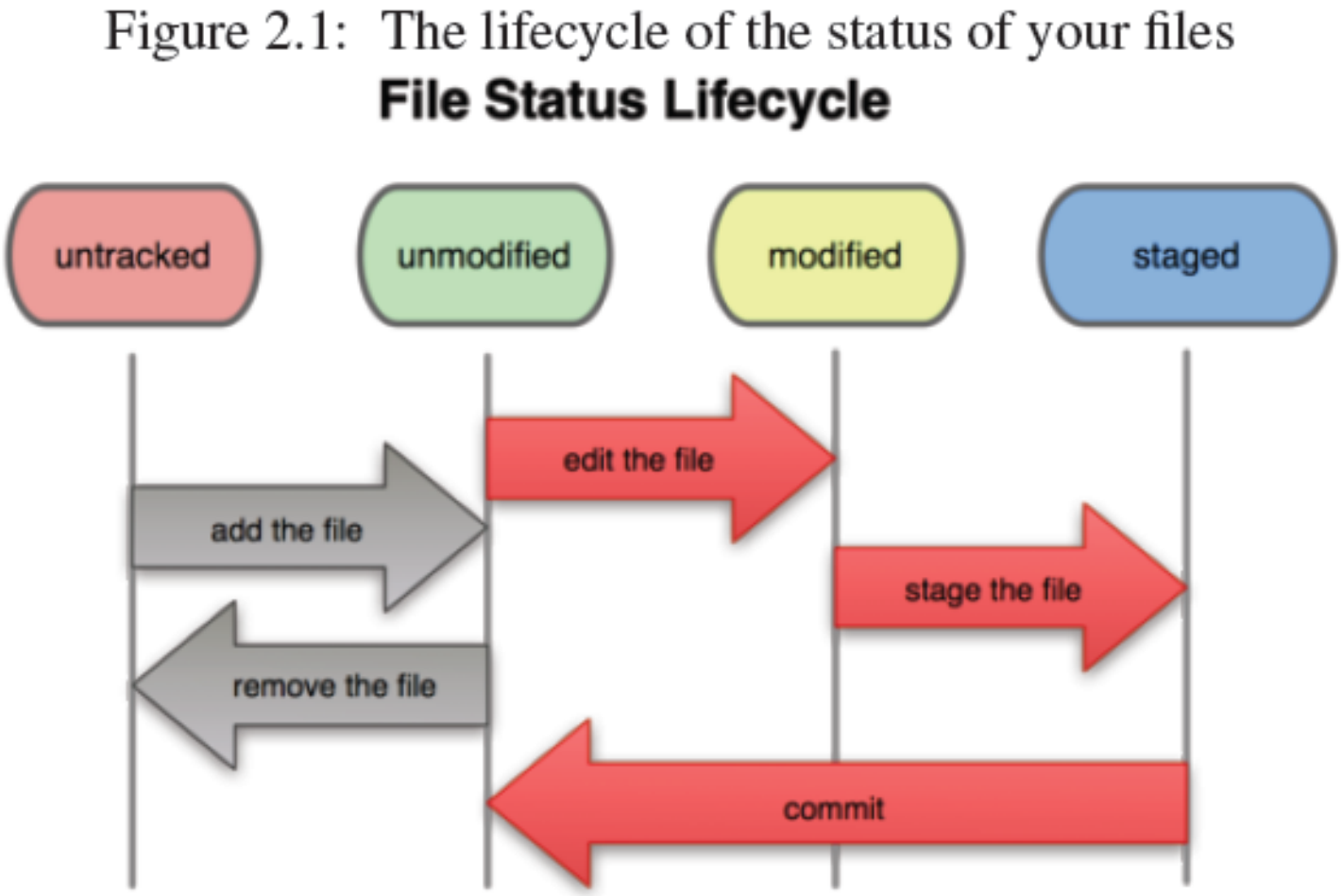
- Creating a local repository
  - Initializing a repository from an existing directory
  - Cloning a remote repository
- Making changes to the repository
  - Tracking new files
  - Staging modified files
  - Committing changes
  - checking the status
  - Viewing the commit history
- Undoing changes

Creating a local repository

- Initializing a repository from an existing directory
  - Go to the project's directory and type git init.
  - This creates a new subdirectory named .git – GIT directory.
- Cloning a remote repository: git clone [url]
  - Creates a new directory
  - Initializes a .git directory in the new directory.
  - Pulls down all the data from the repository.
  - Checks out a working copy of the latest version.
  - Add the repository to your remote server list (named 'origin').
  - Automatically creates a master branch that tracks origin/master.

Making changes

Figure 2.1: The lifecycle of the status of your files



Making changes

- Tracking new files
  - The change will be automatically staged.

    git add README (to begin tracking the README file)
- Staging modified files
  - After you change a file that was already tracked, you need to explicitly stage it into your next commit.

    git add benchmark (assuming that the benchmark file was tracked and modified)
- About git add: a multipurpose command
  - To begin tracking new files
  - To stage files
  - To mark merge-conflicted files as resolved

Making changes

- Checking the status of working directory
  - The git status command determines which files are in which state.

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
# new file:   README
#
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#
# modified:   benchmarks.rb
#
```

Making changes

- Committing changes
  - Only commit the staged changes.

    git commit –m "…"
  - The commit command records the **snapshot** you set up in your staging area.
- Viewing the commit history
  - Use the git log command
  - Many options are available, such as –p, -2, etc.

Undoing changes

- Changing you last commit
  - Overwrite the previous commit, and end up with a single commit.

    git commit --amend
- Unstaging a staged file
  - Cancel the changes made to the staging area.

    git reset HEAD <file>

    E.g. git reset HEAD benchmarks
- Unmodifiying a modified file
  - Cancel the working directory changes

    git checkout -- <file>

    E.g. git checkout -- benchmarks

Working with the Public Repository

- Protocols of accessing remotes
- Creating a repository on the server
- Managing remotes
  - Adding remote repositories
  - Listing remotes
  - Inspecting a remote
  - Removing and renaming remotes
- Fetching and pulling from remotes
- Pushing to remotes

Protocols of accessing remotes

- Local protocol: used when the remote repository is in another directory on disk.

  E.g. git clone /opt/git/project.git
- The SSH protocol: the most common transport protocol of GIT.
  - The only network-based protocol that you can write to.
  - GIT servers authenticate using SSH public keys.
  - E.g. git ssh://user@server:project.git
- The GIT protocol
- The HTTP/S protocol

Creating a repository on a server

- Bare repository: a GIT repository that has no working directory.
- Bare repository is the contents of your project's .git subdirectory. This is your Git repository where all the data of your project snapshots are stored.
- To create a repository from an existing directory of files, simply run git init --bare in that directory.
- An SSH server and a bare repository are the only two things you need to collaborate with other people on a project.

Managing remotes

- Adding remote repositories
  - git remote add [shortname] [url]
  - E.g. git remote add pb git://github.com/paulboone/ticgit.git
- Listing remotes
  - List remote servers that you have configured
  - If you have cloned your repository, you should see at least "origin" – the default name GIT gives to the server you cloned from.
  - git remote –v (the –v option includes the URL that GIT stored for the shortname)

Managing remotes

- Inspecting a remote
  - Shows more information about a particular remote, particularly about the branches.

    git remote show [remote-name]
- Removing and renaming remotes
  - Rename the name of a reference

    git remote rename [old-name] [new-name]
  - Remove a remote

    git remote rm [remote-name]

Fetching and Pulling from Remotes

- Fetching: git fetch [remote-name]
  - Pulls down all the data from the remote project to your local repository.
  - Get references to all the branches from that remote.
  - **Note**: fetching does not make changes to your current working directory.
- Pulling: git pull
  - Running git pull fetches data from the server, and automatically tries to merge it into the code you are currently working on.
  - Can be seen as running git fetch followed by git merge, which will be introduced later.

Pushing to Remotes

- Pushing a local project to the remote (public) repository to share with other people
  - The command: git push [remote] [branch]
  - make your [branch] the new [branch] on the [remote]
  - Requires write access to the remote server
  - If your branch is already on the server, it will try to update it, if it is not, GIT will add it.
  - May be rejected if someone else "pushed" to the server before you.

GIT Branching

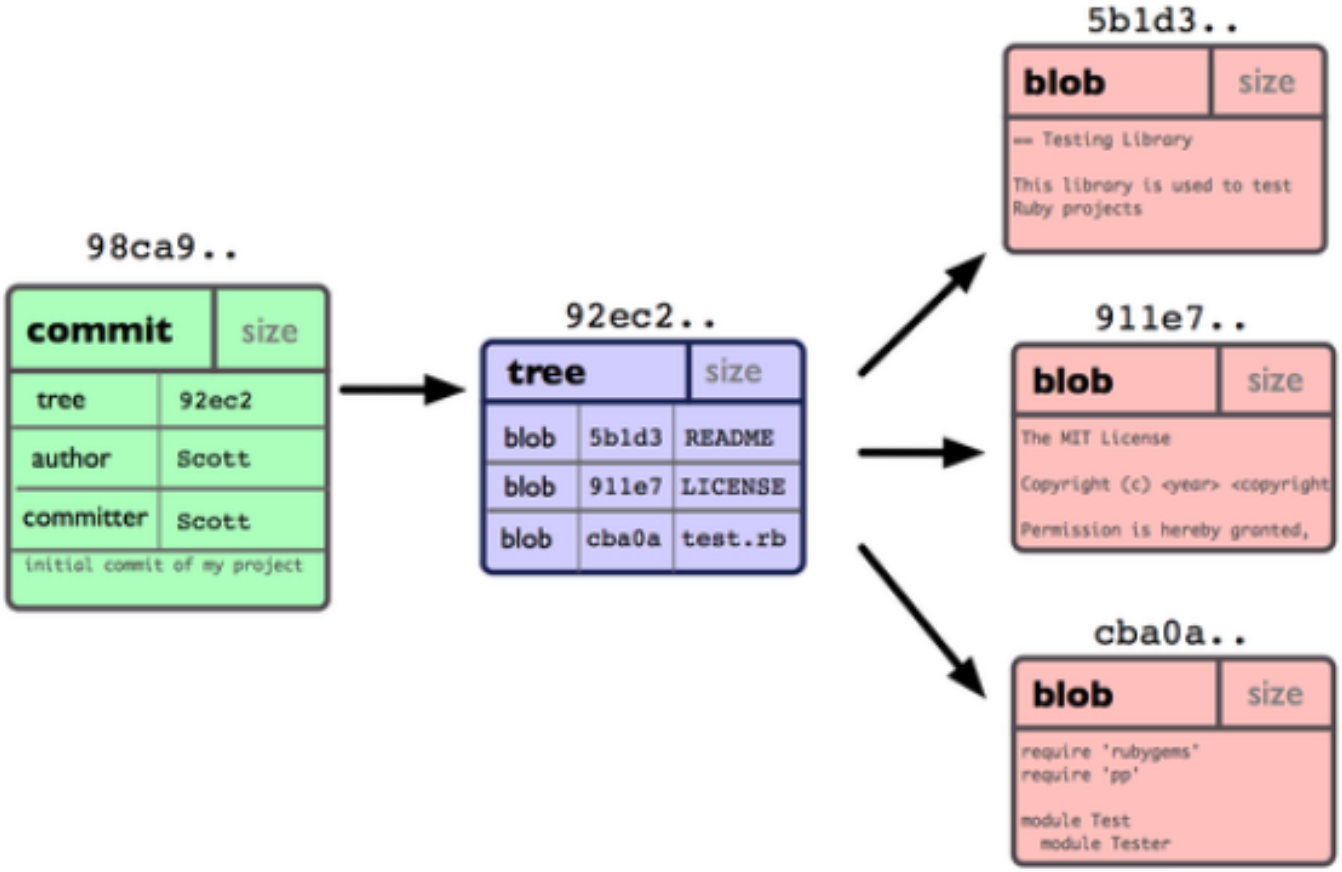
- Branches in GIT
- Basic branching and merging
  - Creating branches
  - Switching between branches
  - Basic Merging
- Remote branches
  - Tracking branches

Branches in GIT

- When you commit in GIT, GIT stores a commit object that contains
  - A pointer to the snapshot of the content you staged.
  - The author and message metadata
  - Zero or more pointers to the commit or commits that were the direct parents of this commit
    - 0 parents: first commit
    - 1 parent: a normal commit
    - >1 parents: a commit resulting from a merge of two or more branches
- A branch in GIT is simply a lightweight movable pointer to one of these commit.
- Creating a new branch in GIT just creates a new pointer.

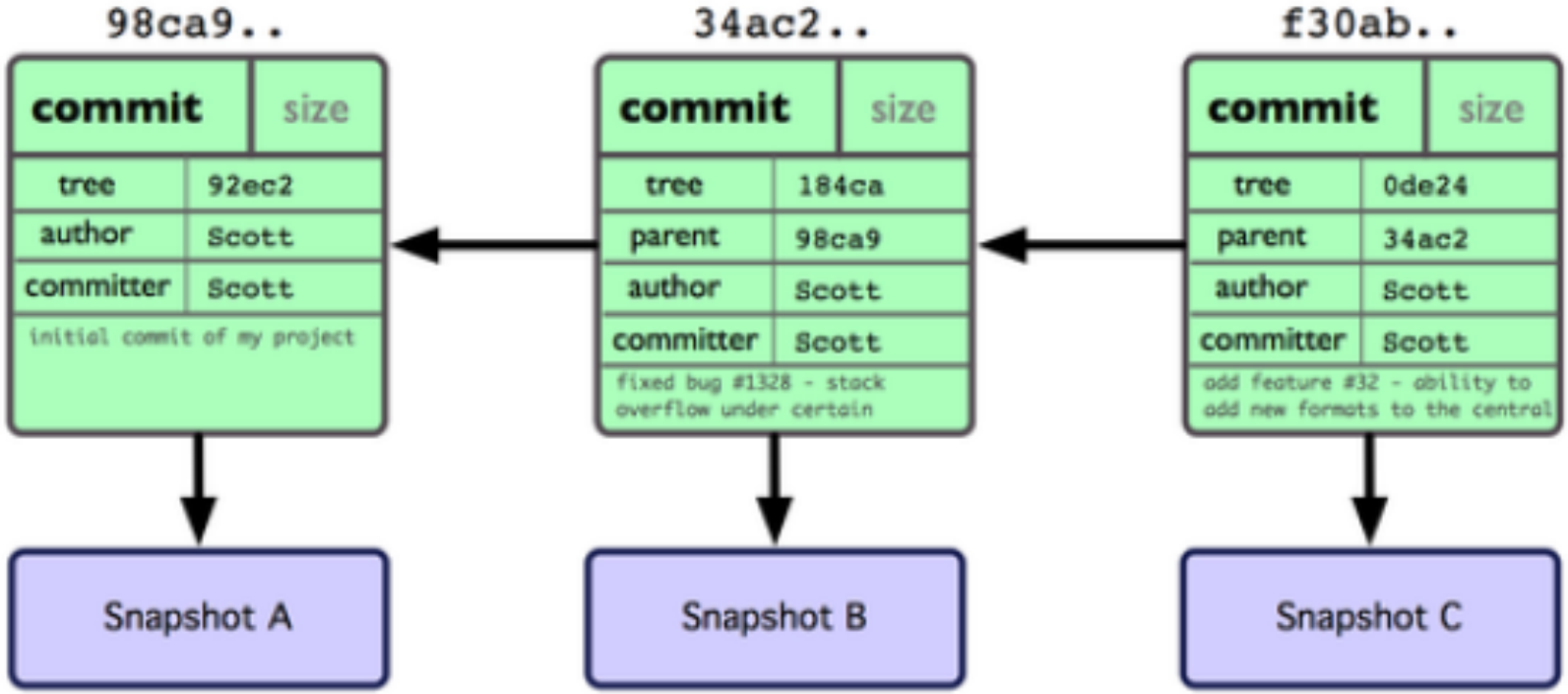
Branches in GIT

Figure 3.1: Single commit repository data



Branches in GIT

Figure 3.2: Git object data for multiple commits



Basic Branching and Merging

- Branch management

  git branch [name-of-the-new-branch] (Create a branch)

  git branch (List all the current branches)

  git branch –d [name-of-the-branch-to-delete] (Delete a branch)
- Switching between branches

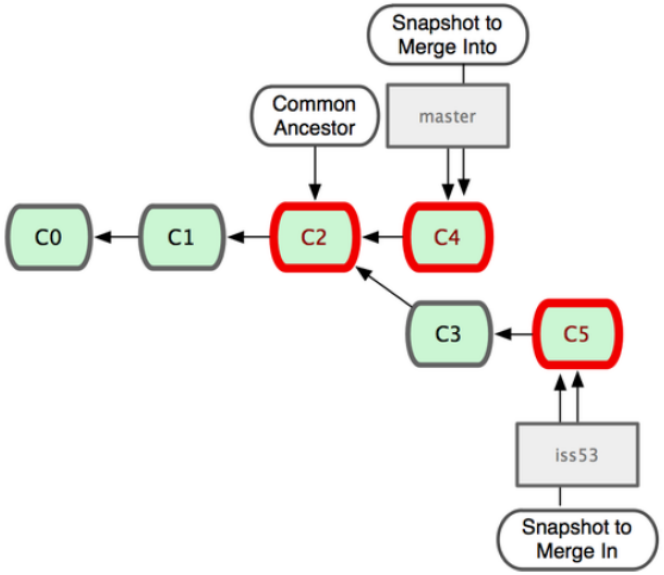
  git checkout [name-of-the-branch]
- Creating a branch and switching to it at the same time

  git checkout –b [name-of-the-branch]

Basic Branching and Merging

- To merge Branch A into Branch B
  - Switch to Branch B: git checkout B
  - Run the git merge command: git merge A
- GIT uses a three-way merging mechanism that identifies the best common ancestor.

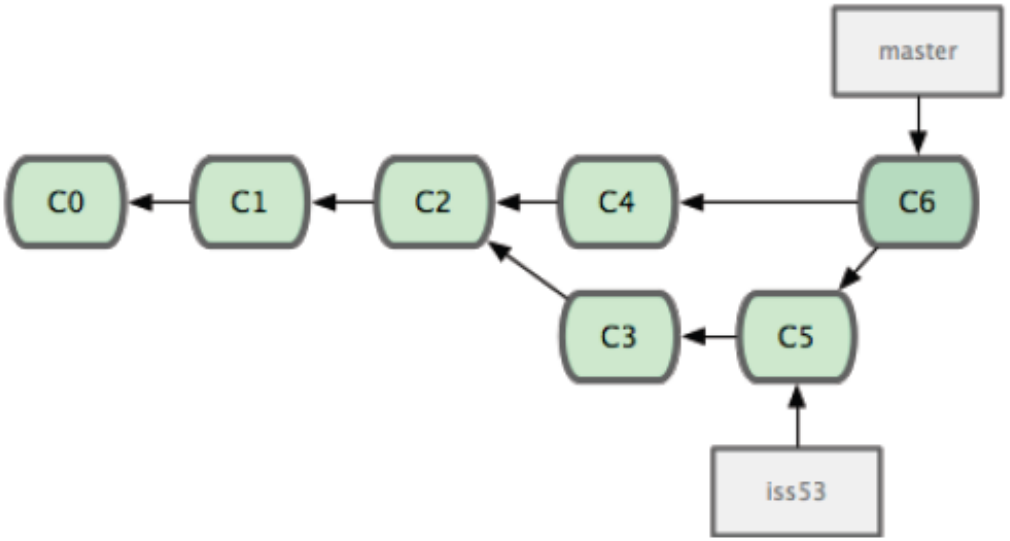
Figure 3.16: Git automatically identifies the best common-ancestor merge base for branch merging.



Basic Branching and Merging

- As the result of merging, GIT creates a new snapshot and automatically creates a new commit that points to it.

Figure 3.17: Git automatically creates a new commit object that contains the merged work.



Basic Branching and Merging

- Merge conflicts: the **same part** of the **same file** was changed **differently** in the two branches you are merging together.

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

- Again, merge conflicts have to be manually resolved.
- After you have resolved the conflicts, run git add on each conflicted file to mark it as resolved.

Remote branches

- Remote branches are **references** to the state of branches on your remote repositories.
- They take the form (remote)/(branch)
  - E.g. origin/master
- Pushing a branch to the server.

  git push (remote) (branch)

  E.g. git push origin serverfix:adifferentname
- Merging a remote branch

  git merge origin/serverfix

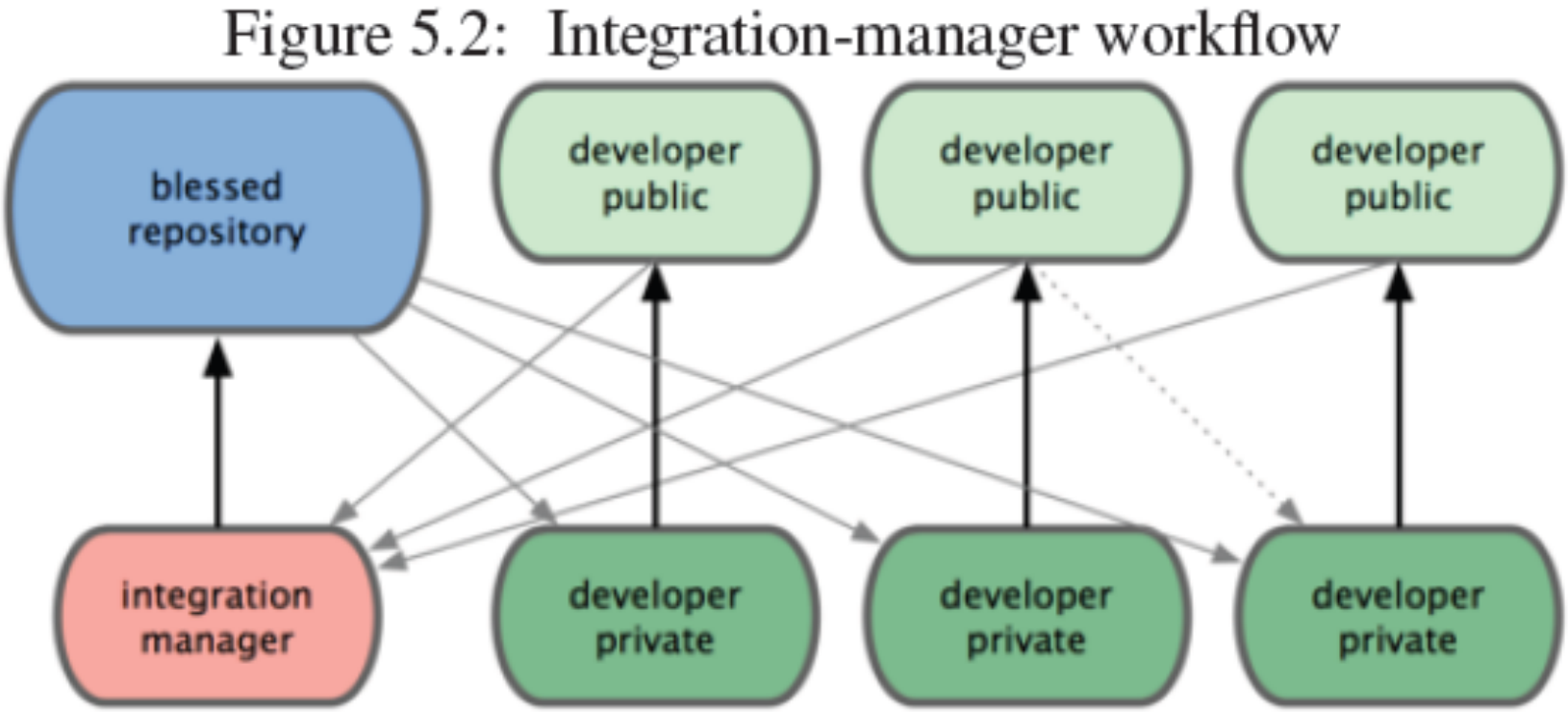
Remote branches

- Tracking branches: checking out a local branch from a remote branch automatically creates what is called a tracking branch.
  - git checkout –b serverfix origin/serverfix
- Tracking branches are local branches that have a direct relationship to a remote branch.
  - If you are on a tracking branch and type git push, GIT automatically knows which server and branch to push to.
  - Similarly, running git pull while on one of these branches fetches all the remote references and then automatically merges in the corresponding branch.

GIT Working Flow

- Having several branches open at the same time, with each used for different stages of your development cycle.
- Centralized workflow
  - All peers share the same public (remote) repository.
- Integration-Manager Workflow
  - There are multiple remote repositories.
  - Each developer has write access to their own public repository, and read access to everyone else's.
  - This scenario often includes a canonical repository that represents the "official project".

GIT Working Flow



Reference

- All the diagrams of this lecture are from the book "Pro Git" by Scott Chacon.