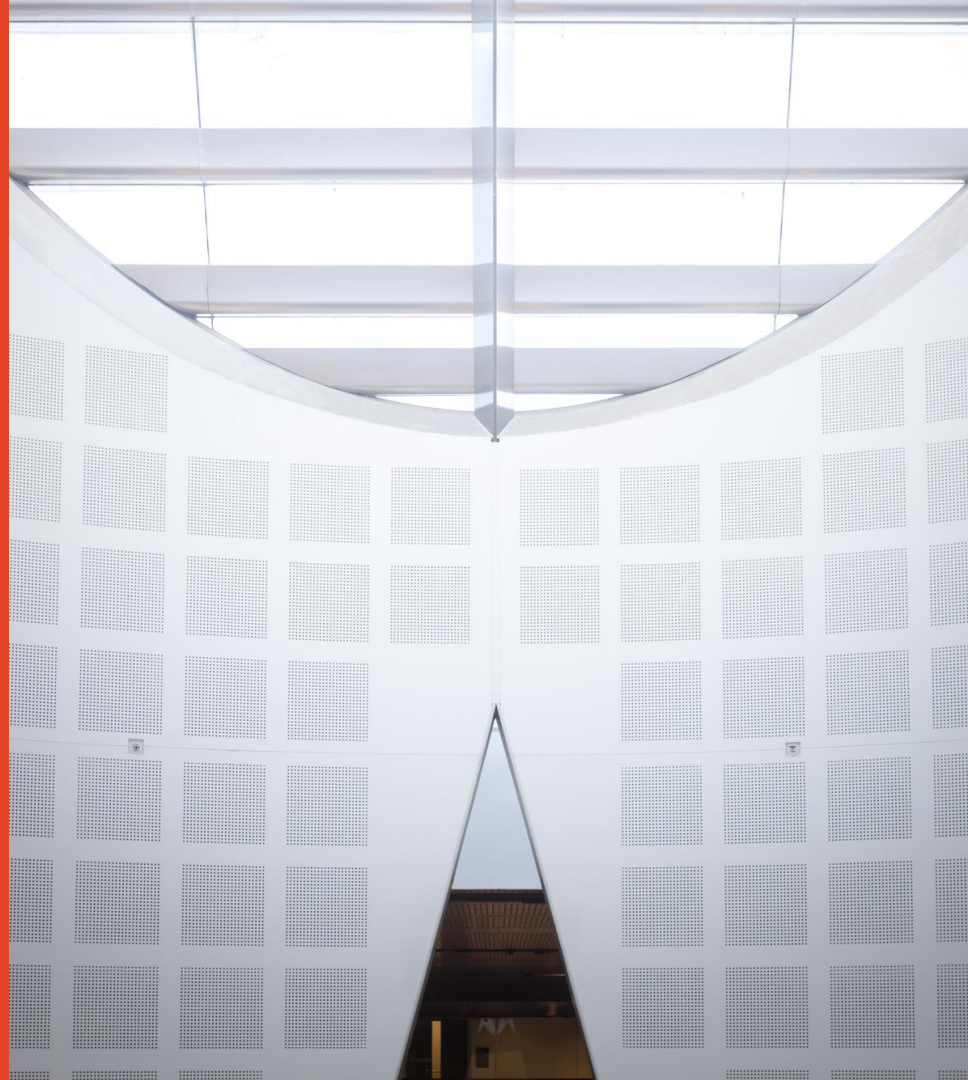


Agile Software Development Practices SOF2412 / COMP9412

Tools and Technologies for
Controlling Artefacts (2)

Dr. Basem Suleiman

School of Information Technologies



Agenda

- Distributed Git
 - Remote Branches
 - Distributed Workflows
 - Collaboration – Workflows
- Working with Repository
 - Own server
 - Hosted service – GitHub

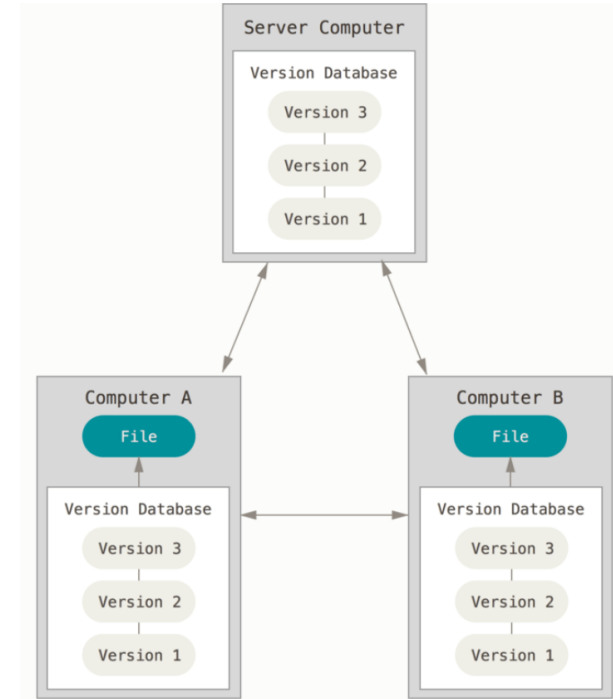
Distributed Git

Remote Branches



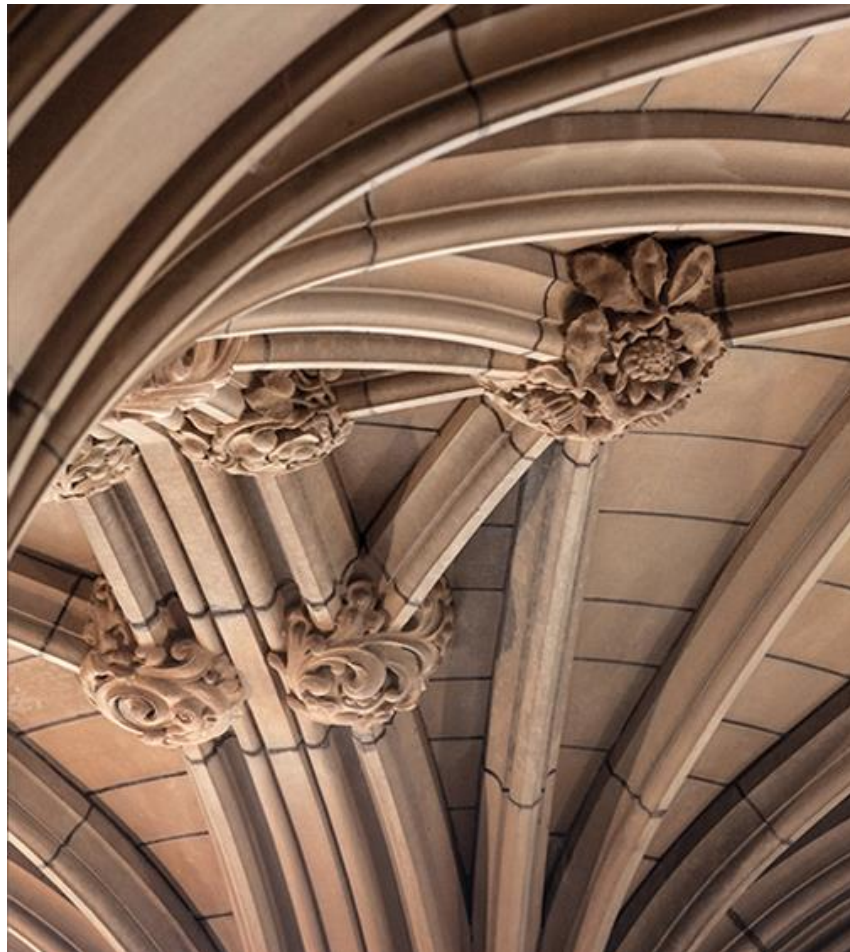
Recall – Distributed Version Control (DVC)

- Developers fully mirror the repository including the full history
- Several remote repositories
 - Developers can collaborate with different groups of people in different ways simultaneously with the same project
 - Can setup several types of workflows (not possible in CVC)



Remote Repository

Running own server




Remote (Hosted) Repository

- A **remote repository** is generally a **simple repository** – the contents of your project's .git directory and nothing else
- When you really need to work with remote repository?
- **One-person project**
 - **Local repository should suffice** – includes working directory
 - Track changes and history of development as individual
- **Team-based (collaboration) projects**
 - **Remote repo team members (collaborators) can access anytime**
 - **More reliable common repo (rather own local repo)**
 - **All team members can push and pull**
 - **Need to have some coordination and permission control**

Remote Branches

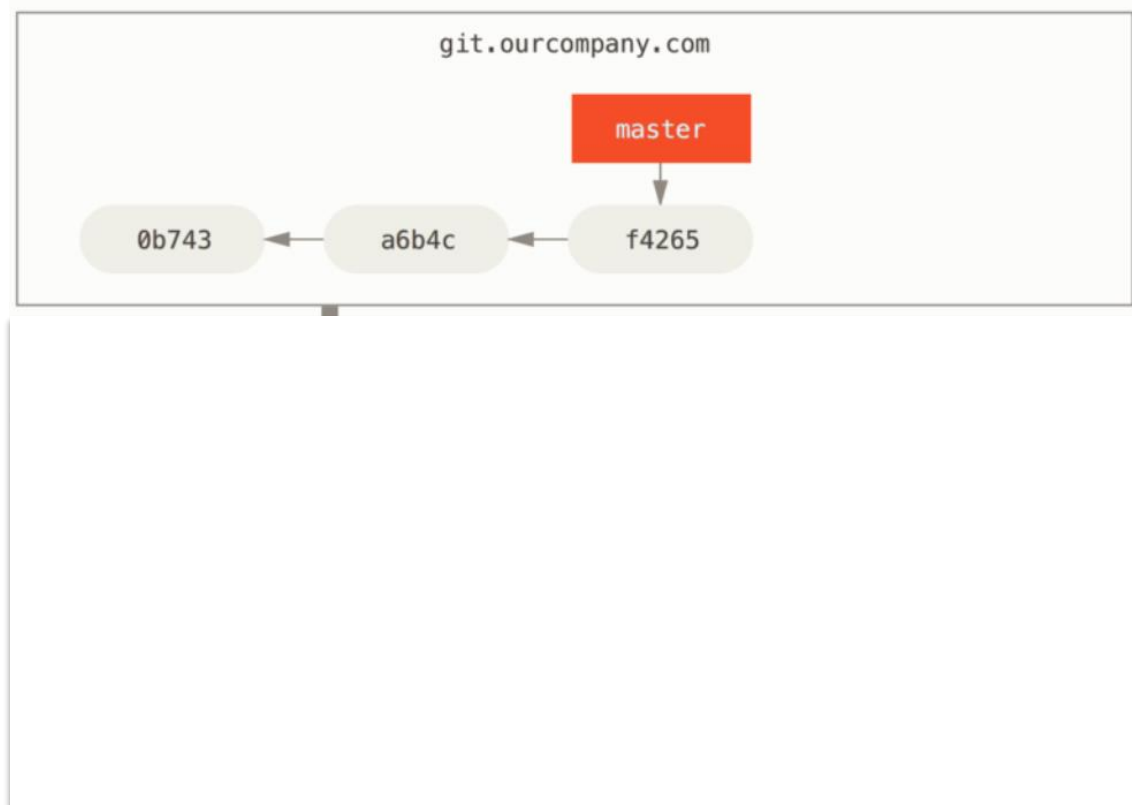
- **Remote references:** references (pointers) in your remote repos.
 - *git ls-remote [remote]*: get full list of remote references
 - *git remote show [remote]*: get list of remote branches
- **Remote-tracking branches:** references to the state of remote branches
 - Local references you cannot move; git moves them for you to make sure they accurately represent the state of the remote repo
 - Form: *<remote> / <branch>*
 - E.g., check the origin/master branch to see the master branch on your origin remote look like

Remote-Tracking Branches – Example

- You have a git server on your network (*git.ourcompany.com*)
- *git clone git.ourcompany.com* will:
 - Names it **origin** 
 - Pulls down all its data
 - Creates a pointer to where its master branch is and call it origin/master locally
 - Set your own local master branch starting at the same place as origin's master branch
- **Note:** origin is the default name for a remote when you run git clone
 - *git clone -o MyBranch* to name your default remote branch MyBranch/master

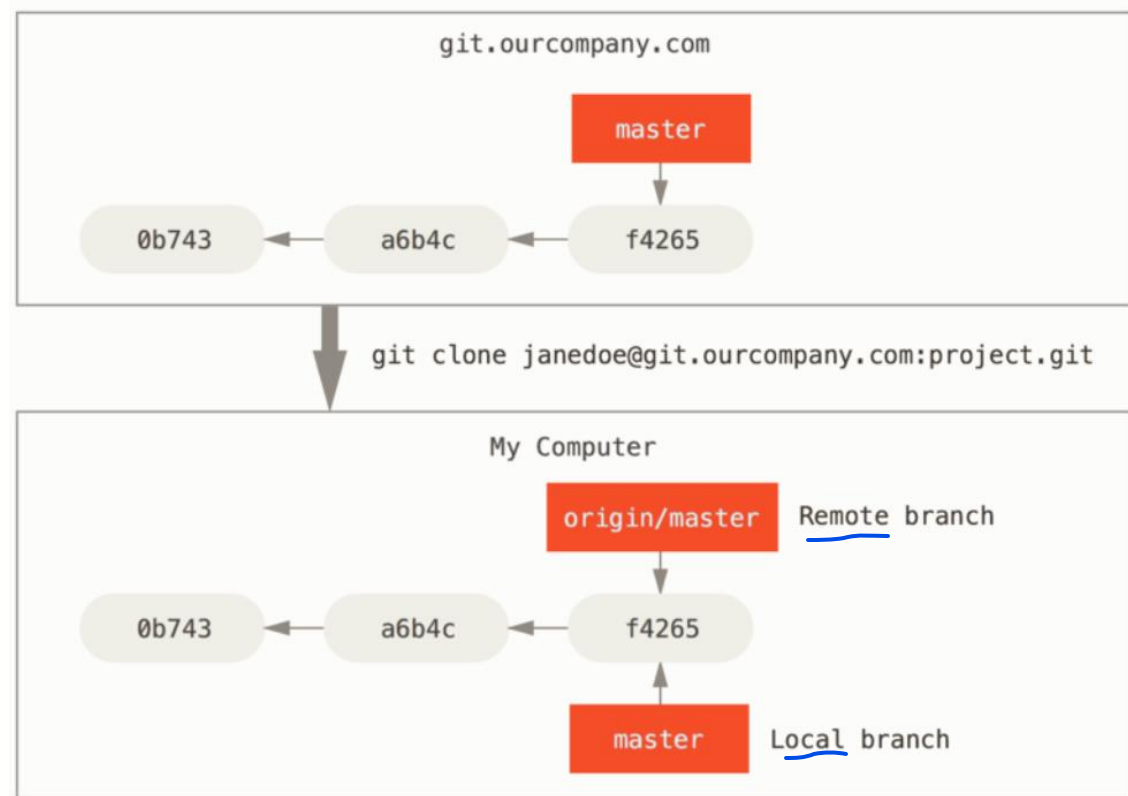
Remote-Tracking Branches – Clone Remote Repo

git server (remote repo.)



Remote-Tracking Branches – Clone Remote Repo

git server (remote repo.)

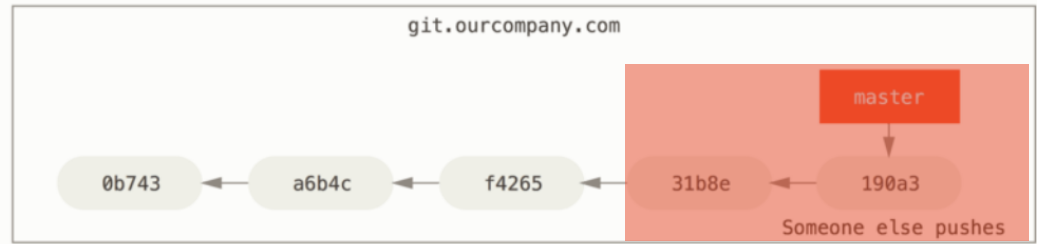


Your PC (local repo.)
git clone ...

Local and Remote Branches

- Imagine you do some work on your local branch, while another developer pushes updates to the master branch of `git.ourcompany.com`?

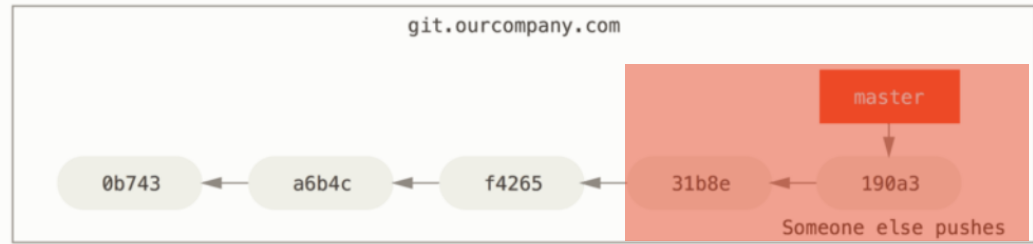
Developer's updates
pushed to the remote



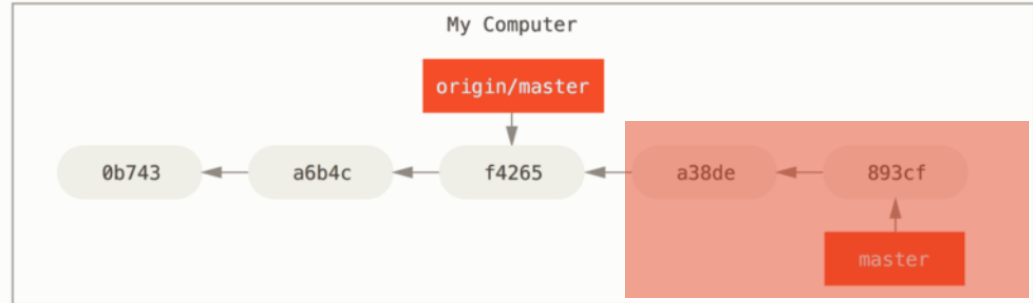
Local and Remote Branches

- Imagine you do some work on your local branch, while another developer pushes updates to the master branch of git.ourcompany.com?

Developer's updates
pushed to the remote



Your local updates

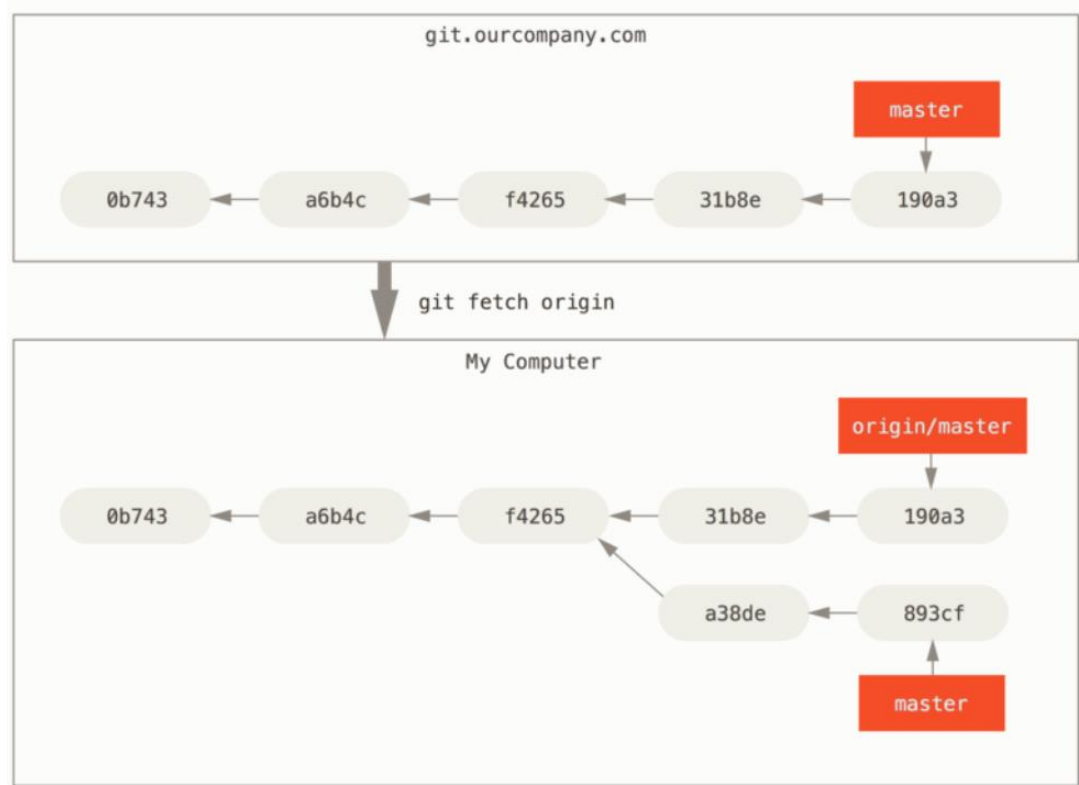


Local and Remote Branches – Synchronization

To sync. your work run:

git fetch origin

Fetches changes you do not have from the remote
and update your local
repo – moving your
origin/master pointer to its
new (up-to-date) position



Remote-Tracking Branches – Pushing

- To share local branch, explicitly push it to a remote you have write access to

git push origin servfix

“Take my servfix local branch and push it to update the remote’s servfix branch.”

git push origin servfix:servfix

“Take my servfix and make it the remote’s servfix”

Remote-Tracking Branches – Pushing

- A collaborator wants to fetch *serverfix* from the remote

git fetch origin

- They get a reference to where the server's version of *serverfix* is under the remote branch *origin/serverfix*
 - They only have an *origin/serverfix* pointer that they can't modify
- How you can merge this into your current working branch?

Remote-Tracking Branches – Merge/Base

- To merge this work into your working branch:

`git checkout -b serverfix origin/serverfix`

```
Branch serverfix set up to track remote branch serverfix from origin.  
Switched to a new branch 'serverfix'
```

- To work on your own serverfix branch you can base it off your remote-tracking branch
- This gives you a local branch that you can work on that starts where origin/serverfix is

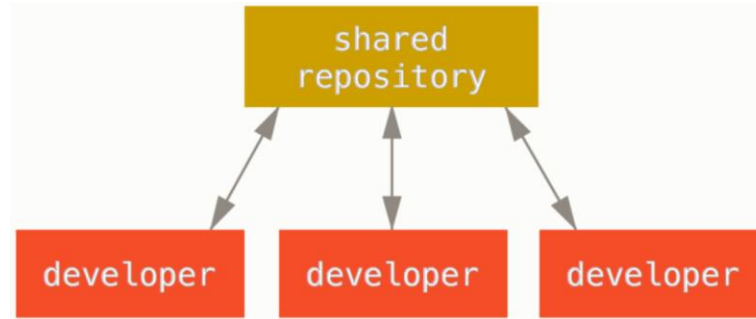
Distributed Git

Distributed workflows



Centralized VCSs

- *Single collaboration model* (centralized workflow)
 - Every **developer is a node** working **on a central shared repo.** and sync. to it



- Not limited to small teams; git **branching** allows 100's of developers to work on a single **project** through many branches **simultaneously**
- If suitable, create a repo. and give every developer ***push*** access

Centralized VCSs – Workflow

- In a centralized VCS model, Joe and Sarah clone from a shared repo. and both make changes to some files locally

Centralized VCSs – Workflow

- In a centralized VCS model, Joe and Sarah clone from a shared repo. and both make changes to some files locally
- Discuss:
 - What happens when Joe pushes his changes to the repo. first?
 - What happens when Sarah pushes her changes after Joe?

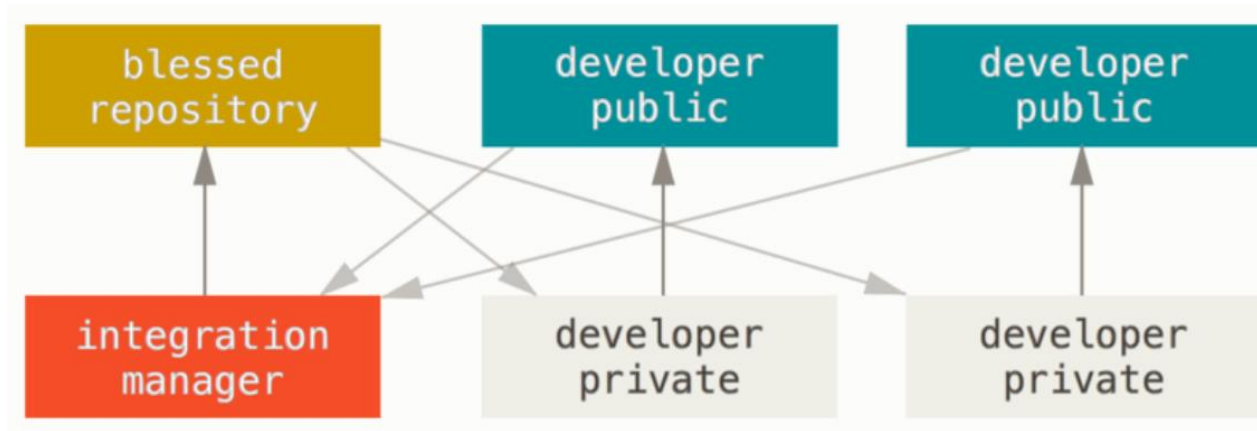
1) Centralized VCSs – Workflow

- In a centralized VCS model, Joe and Sarah clone from a shared repo. and both make changes to some files locally
- Discuss:
 - What happens when Joe pushes his changes to the repo. first?
 - What happens when Sarah pushes her changes after Joe?
 - The server will reject the changes.
 - Sarah must first fetch the Joe's changes from the server and merge it locally before pushing the merged changes

2) Distributed VCS

- Git allows every developer to be both
 - Node: can contribute code to other repos.
 - Shared repo.: maintain a public repo. on which others can base their work and which they can contribute to
- Allows wide range of workflow possibilities for projects/teams
- Discuss common designs and discuss pros and cons of each

Distributed VCS – Integration-Manager Model



Integration-Manager Model (permission model)

- Often includes a canonical repo. that represents the “official” project
 - Each developer has write-access to their own public repo. and read-access to everyone else’s
 - Developers make their own public clone of the project and push their changes to it
 - Then they inform the maintainer of the main project pull their changes
 - The maintainer add developer’s repo as a remote, test changes locally merge them into the branch and push back to the their repo

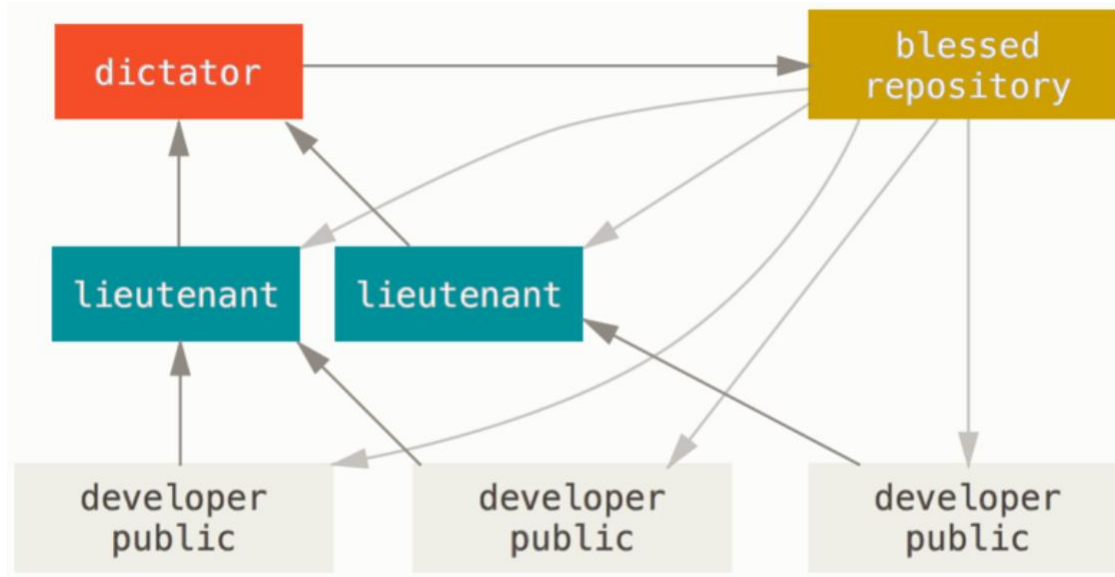
Integration-Manager – Workflow

1. The project maintainer pushes to their public repository.
2. A contributor clones that repository and makes changes.
3. The contributor pushes to their own public copy.
4. The contributor sends the maintainer an email asking them to pull changes.
5. The maintainer adds the contributor's repository as a remote and merges locally
6. The maintainer pushes merged changes to the main (blessed) repository.

Integration-Manager – Use

- Very common workflow in hosted servers such as GitHub and GitLab
- Easy to fork a project and push your changes into your fork for everyone to see
- Developers can continue to work on their repos. while the maintainer of the main repo. can pull their changes at anytime
- Contributors do not have to wait for the project to incorporate their changes
 - each can work on their pace

Distributed VCS – Dictator and Lieutenants



Dictator and Lieutenants Model

- Variation of multiple-repository workflow
- **Lieutenants** various integration managers are in charge of certain parts of the repo.
- **Benevolent dictator** All Lieutenants have one integration manager
- The benevolent dictator pushes from his directory to a reference repository from which all the collaborators need to pull

Dictator and Lieutenants - Workflow

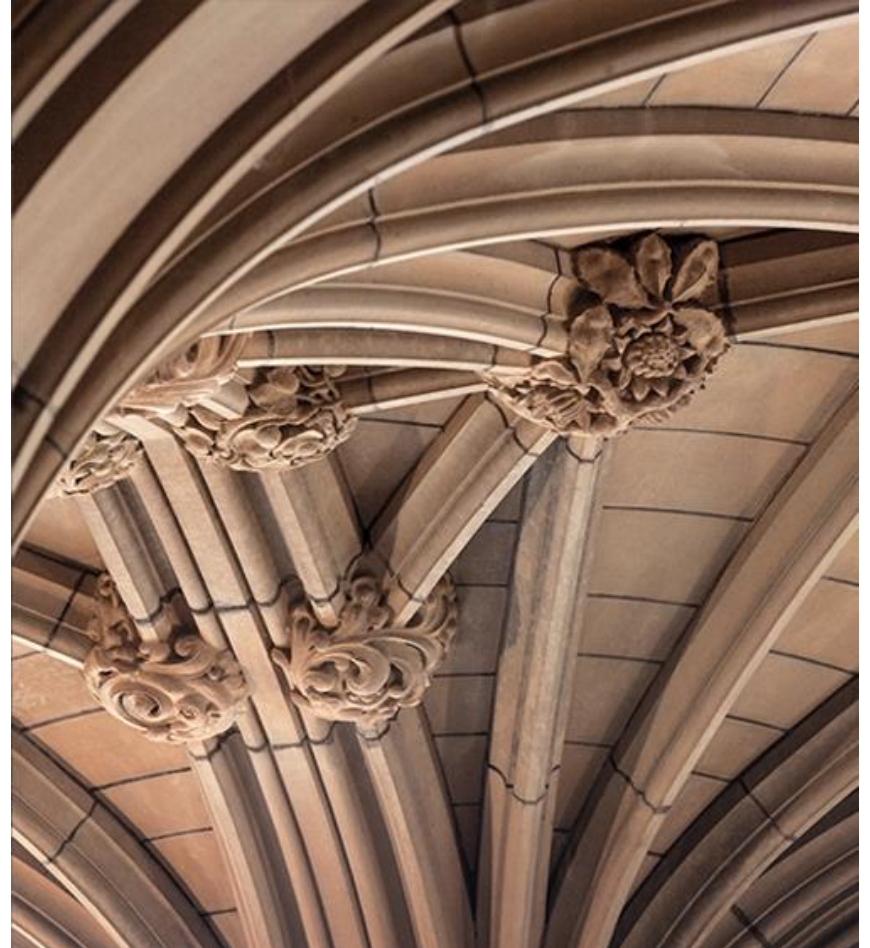
- Regular developers work on their topic branch and rebase their work on top of master. The master branch is that of the reference repository to which the dictator pushes
- Lieutenants merge the developers' topic branches into their master branch.
- The dictator merges the lieutenants' master branches into the dictator's master branch.
- Finally, the dictator pushes that master branch to the reference repository so the other developers can rebase on it.

Dictator and Lieutenants – Use

- For very big projects or in highly hierarchical environments
 - Hundreds of collaborators, e.g., Linux Kernel ←
- Project leader (the dictator) to delegate much of the work and collect large subsets of the code at multiple points before integrating them

Contributing to a Project

Centralized workflow



Contributing to a Project (1)

- Teams can contribute to a git project in various ways (as git is flexible)
- Factors affect how one can contribute effectively to a project

1. Active contributor count: how many users are actively contributing code to this project, and how often?

- e.g., 2-3 developers with a few commits a day
- E.g., 100's of developers with 100's commits daily
- What is the relationship between number of developers and commits? commits and potential conflict/merge issues?

Contributing to a Project (2)

2. Project workflow:

- Centralized with equal write access to main code-line?
- Does the project have a maintainer or integration manager who check all commits?
- Is a lieutenant system in place and do you have to submit your work to them first?

3. Commit access:

- Do you have write-access?
- If not, is there a policy on how the contributed work is accepted?
- How much work a developer may contribute at a time? and how often?

Contributing to a Project (3) – Commit Guidelines

- No whitespace errors:
 - Whitespace errors: change that introduces a trailing whitespace, whitespace-only line or tab
- run git diff --check before commits to identify and list possible whitespace errors
- Alternatively, configure git to ignore the warning
 - git config apply.whitespace nowarn

Contributing to a Project (3) – Commit Guidelines

- **Commit logically separate changeset:** do not work on many different issues in your code and submit them as one commit!
- **Use quality commit messages:** a concise description of the change followed by a blank line then a detailed explanation
 - Check this [note about git commit messages by Tim Pope](#)
- More guidelines: git has a full guide for commits described in [Git source code](#)

Contributing to a Private Small Project (1)

- Private project with few developers all have push access to the repo
- Centralized workflow with offline committing and simple branching and merging

⇒ Scenario: 2 developers working on a shared repo.

- John clones the repo., make a change and commits locally
- Jessica clones the repo., make a change and commits locally
- Jessica punches her work to the server, and this should work fine
- shortly afterwards, John makes some changes, commits them to his local repository, and tries to push them to the same server
- John's push fails because of Jessica's earlier push of her changes

Contributing to a Private Small Project (2)

```
# John's Machine
$ git clone john@github:simplegit.git
Cloning into 'simplegit'...
...
$ cd simplegit/
$ vim lib/simplegit.rb
$ git commit -am 'remove invalid default value'
[master 738ee87] remove invalid default value
1 files changed, 1 insertions(+), 1 deletions(-)
```

1

```
# Jessica's Machine
$ git clone jessica@github:simplegit.git
Cloning into 'simplegit'...
...
$ cd simplegit/
$ vim TODO
$ git commit -am 'add reset task'
[master fbff5bc] add reset task
1 files changed, 1 insertions(+), 0 deletions(-)
```

2

```
# Jessica's Machine
$ git push origin master
...
To jessica@github:simplegit.git
1edee6b..fbff5bc master -> master
```

3

```
# John's Machine
$ git push origin master
To john@github:simplegit.git
! [rejected] master -> master (non-fast forward)
error: failed to push some refs to 'john@github:simplegit.git'
```

4

Contributing to a Private Small Project (3)

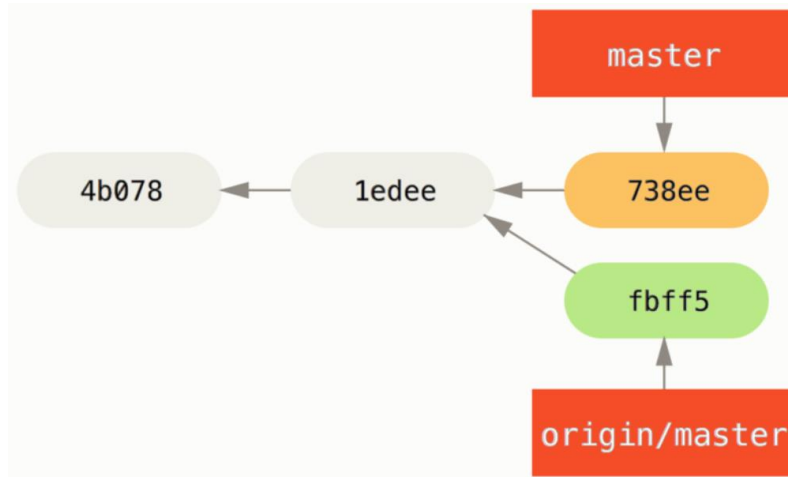
- John fetches Jessica's

```
$ git fetch origin
```

```
...
```

```
From john@github:simplegit
```

```
+ 049d078...fbff5bc master -> origin/master
```



Contributing to Private Small Project (4)

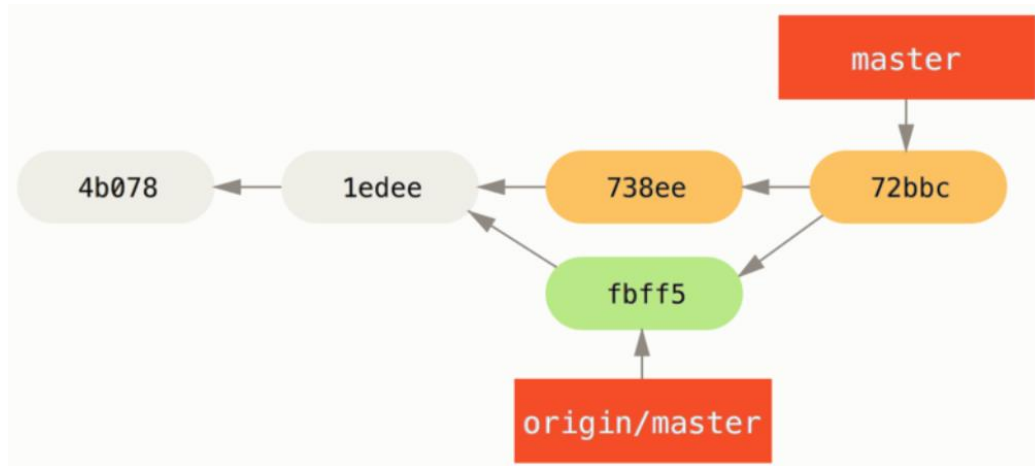
- Now John can merge Jessica's work that he fetched into his own local work:

```
$ git merge origin/master
```

```
Merge made by the 'recursive' strategy.
```

```
TODO | 1 +
```

```
1 files changed, 1 insertions(+), 0 deletions(-)
```



Contributing to Private Small Project (5)

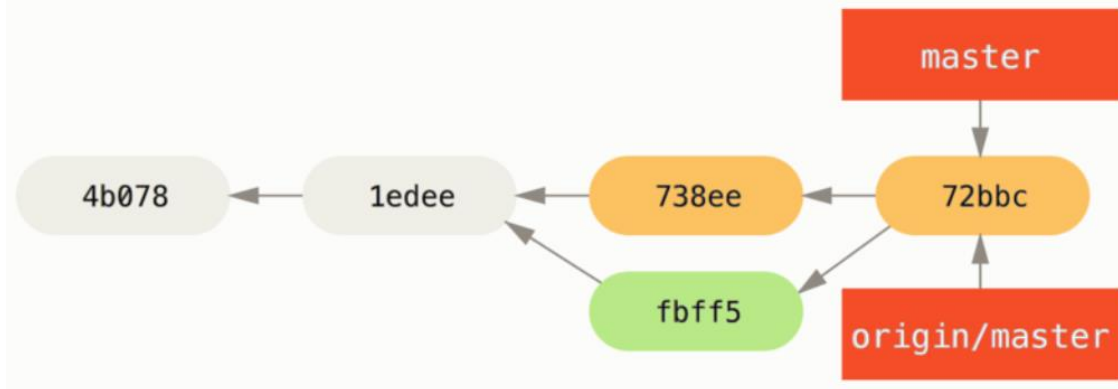
- John tests this new code to make sure none of Jessica's work affects any of his and, he can finally push the new merged work up to the server

```
$ git push origin master
```

```
...
```

```
To john@githost:simplegit.git
```

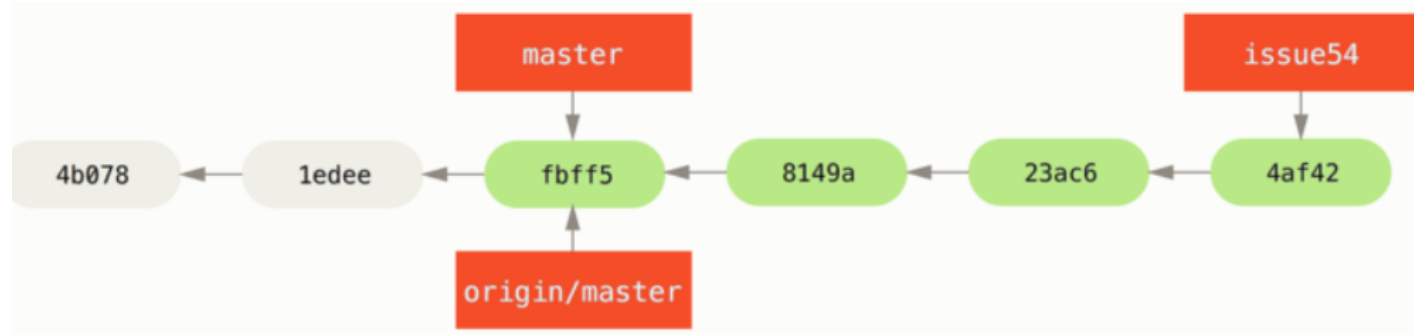
```
fbff5bc..72bbc59 master -> master
```



Contributing to Private Small Project (6)

Meanwhile, Jessica created a new topic branch *issue54*, and made three commits to that branch

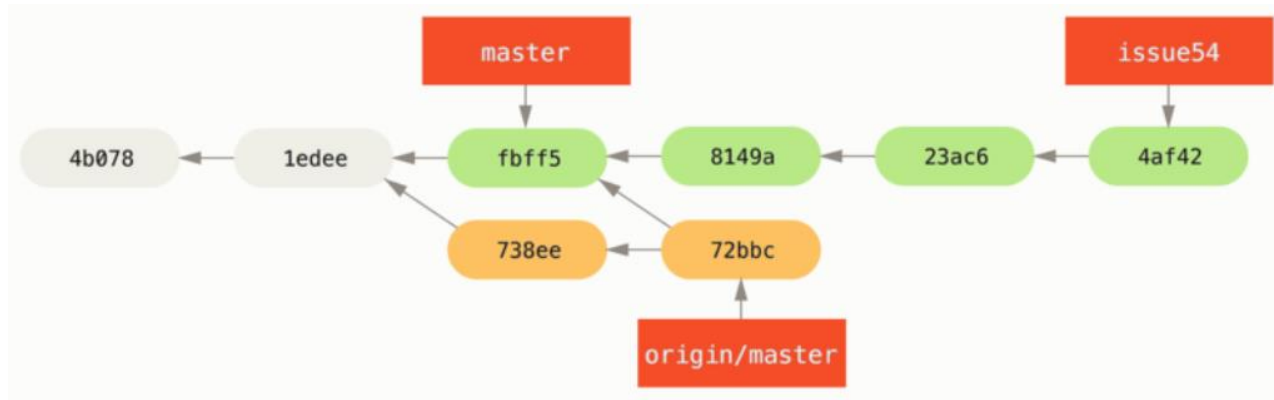
She hasn't fetched John's changes yet, so her commit history looks like this



Contributing to Private Small Project (7)

Jessica wants to get John's new work from the repo. and examine it:

```
# Jessica's Machine
$ git fetch origin
...
From jessica@githost:simplegit
    fbff5bc..72bbc59  master    -> origin/master
```



Contributing to Private Small Project (8)

Jessica thinks her topic branch is ready, but she wants to know what part of John's fetched work she has to merge into her work so that she can push

```
$ git log --no-merges issue54..origin/master
commit 738ee872852dfaa9d6634e0dea7a324040193016
Author: John Smith <jsmith@example.com>
Date:   Fri May 29 16:01:27 2009 -0700
```

```
remove invalid default value
```

issue54..origin/master is a **log filter** that asks git to display only those commits that are on origin/master branch that are not on the case issue54

The output tells there is a single commit that John has made that Jessica has not merged into her local work.

If she merges origin/master, that is the single commit that will modify her local work.

Contributing to Private Small Project (9)

Now, Jessica can merge her topic work into her master branch, merge John's work (origin/master) into her master branch, and then push back to the server again

```
$ git checkout master
```

```
Switched to branch 'master'
```

```
Your branch is behind 'origin/master' by 2 commits, and can be fast-forwarded.
```

Contributing to Private Small Project (9)

Jessica can merge either ¹ origin/master or ² issue54 first — they're both upstream, so the order doesn't matter

```
$ git merge issue54
Updating fbff5bc..4af4298
Fast forward
 README      |    1 +
 lib/simplegit.rb |    6 +++++
 2 files changed, 6 insertions(+), 1 deletions(-)
```

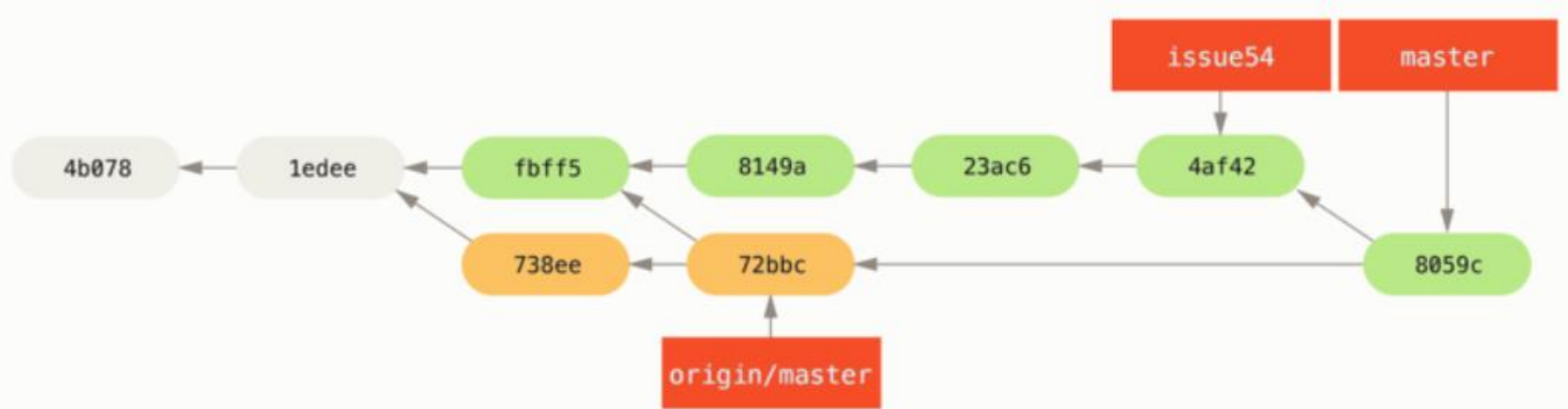
No problems occur; as you can see it was a simple fast-forward merge.

Contributing to Private Small Project (10)

Jessica now completes the local merging process by merging John's earlier fetched work that is sitting in the origin/master branch:

```
$ git merge origin/master
Auto-merging lib/simplegit.rb
Merge made by the 'recursive' strategy.
 lib/simplegit.rb | 2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)
```

Everything merges cleanly, and Jessica's history now looks like this:

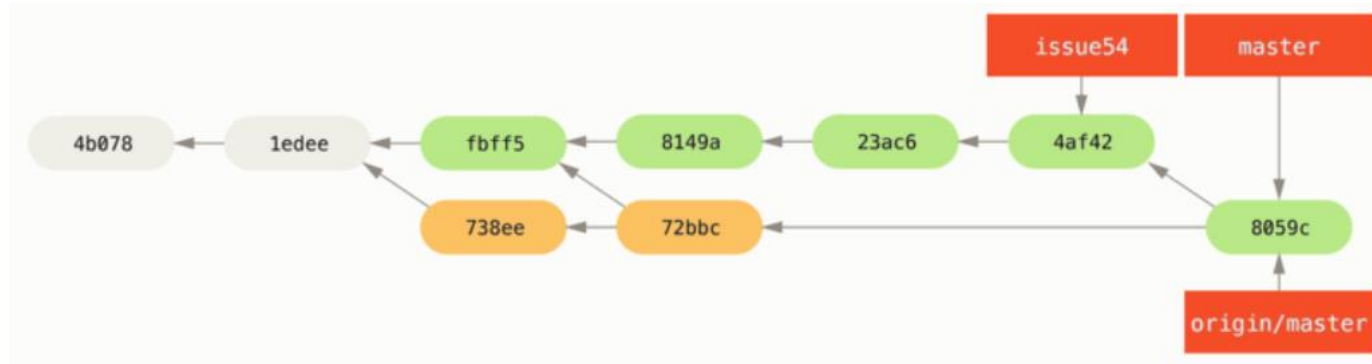


Contributing to Private Small Project (11)

Now origin/master is reachable from
Jessica's master branch, so she should be able to
successfully push:

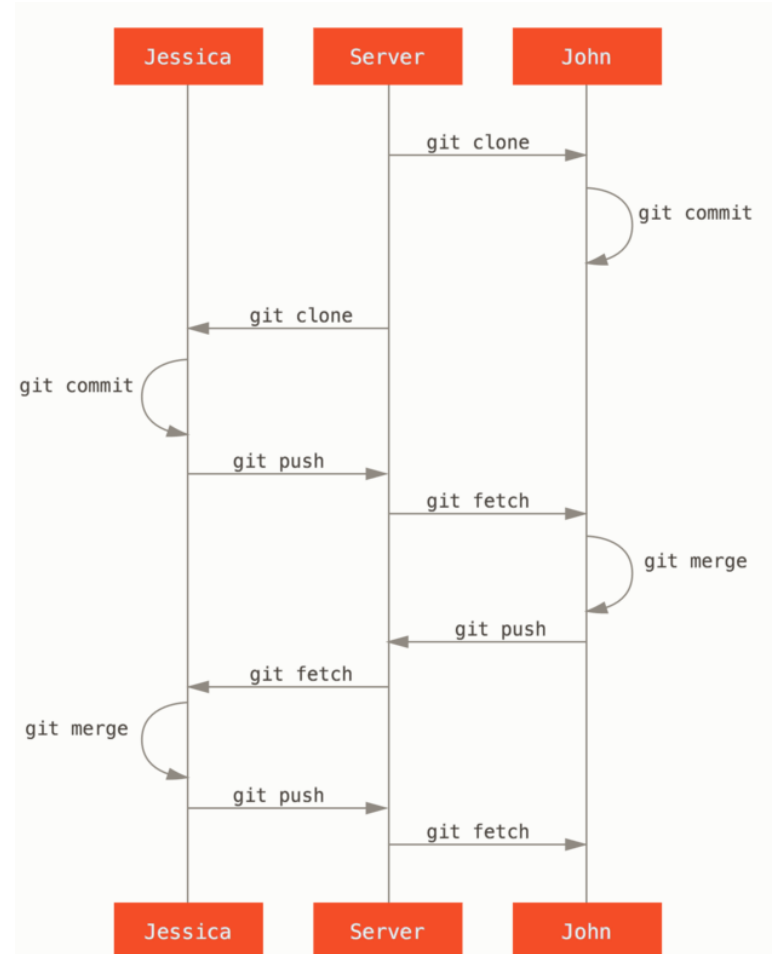
```
$ git push origin master
...
To jessica@github:simplegit.git
72bbc59..8059c15  master -> master
```

Jessica and John has committed a few times and merged each other's work successfully.



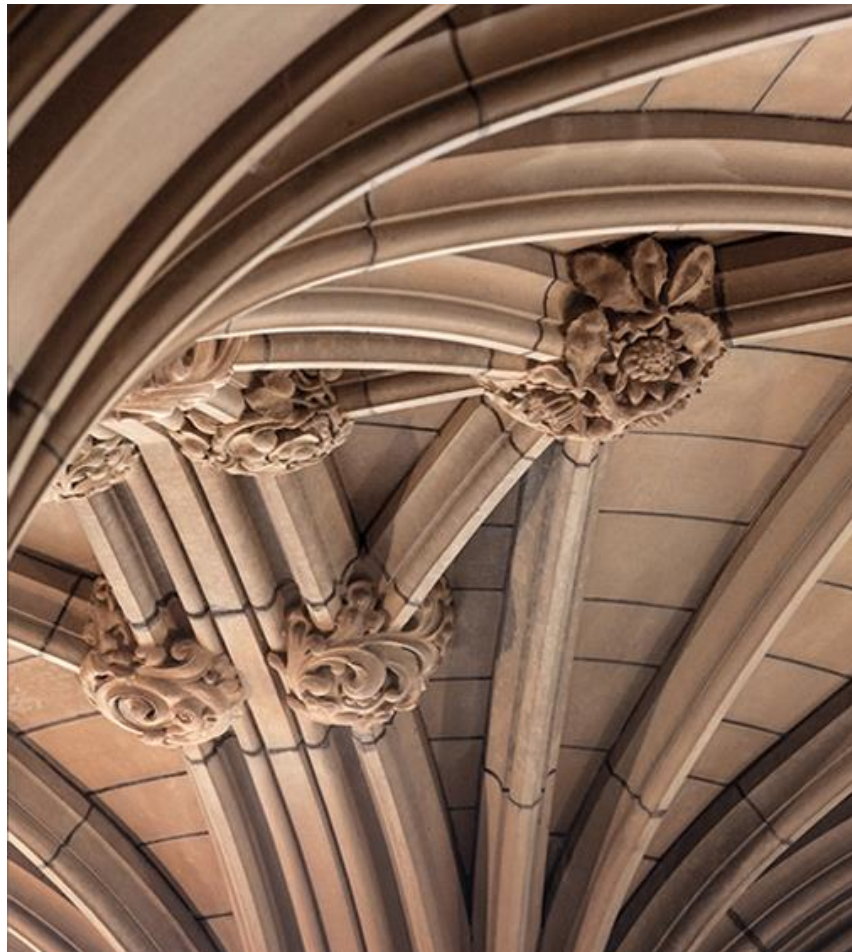
Contributing to Private Small Project (12)

General sequence of events for a simple multiple-developer git workflow



Remote Repository

Running own server



⇒ Remote Repository – Running Own Server

- Hosting our code/projects on your own server
 - Configure which protocols your server to communicate with
 - Typical server set-ups using the configured protocols

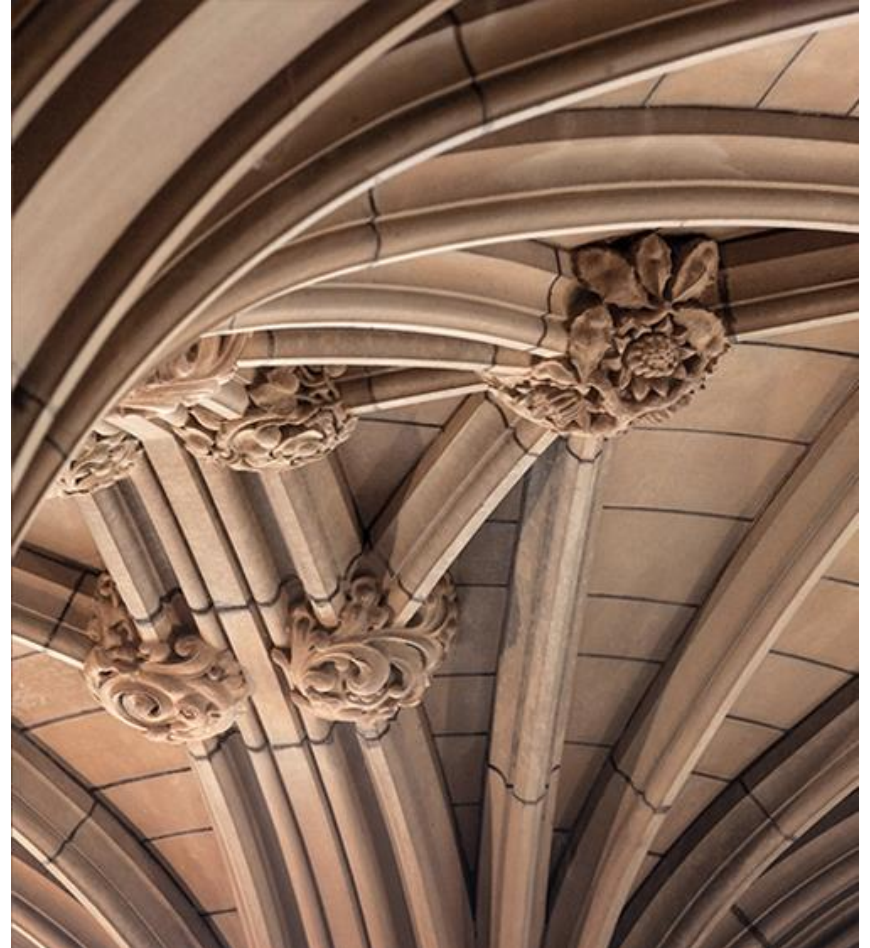
Protocol	Pros	Cons
File system	simple, support access control	public share is difficult to setup
SSH	easy to setup (most systems provide ssh tools), fast (compress data), support authenticated write access	no anonymous access (even read access)
HTTP	unlikely to be blocked	Can become difficult to setup
Git	Fastest protocol, allow anonymous public access	Difficult to setup, lack of authentication, use non standard port (9418) which can be blocked

⇒ Remote Repository – Running Hosted Server²

- Set-up your project/code directory on a hosted server (Git server)
 - No concerns about security or privacy
 - Avoid the hassle of setting and maintaining your own server
- Many hosting services including GitHub, GitLab, BitBucket, Mercurial
 - Not Git itself but a hosting service for Git repos
 - Host your own projects and open it up for collaboration
 - Create organization, teams and repos
 - Web-based and desktop/command-line interactions
 - **Public Repos/projects**
 - **Private Repos/projects**

Remote Repository

Hosted service – GitHub



Hosted Servers – GitHub

- There are large number of Git hosting options
 - We will focus on Github as it is the largest Git host
- Create one-user (personal) account
- Public and private repos

GitHub – Organizations

- Allows collaboration across many projects at the same time in organization
 - Group of people with shared ownership of projects
- Organization's members roles:
 - **Owner:** have complete administrative access to the organization
 - **Member:** everyone else
- **Owners** can **manage members' access** to the organization's repos. and projects **with fine-grained permission controls**
 - Create your own organization
 - Understand and carefully manage members access to your organization
- How about external collaborators (consultant) ?

GitHub – Organization Access Control

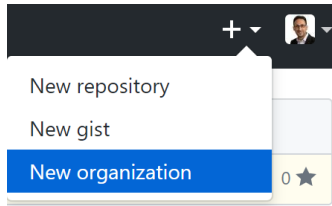
Organization action	Owners	Members
Invite people to join the organization	X	
Edit and cancel invitations to join the organization	X	
Remove members from the organization	X	
Reinstate former members to the organization	X	
Add and remove people from all teams	X	
Promote organization members to <i>team maintainer</i>	X	
Add collaborators to all repositories	X	
Access the organization audit log	X	
Delete all teams	X	
Delete the organization account, including all repositories	X	

Organization action	Owners	Members
Create teams	X	X
See all organization members and teams	X	X
@mention any visible team	X	X
Can be made a <i>team maintainer</i>	X	X
Transfer repositories	X	
View a project board and add or reorganize its cards and columns	X	X
Create or delete a project board and edit its description	X	X
Automate actions for project boards	X	X
View and post private team discussions to all teams (see " About team discussions " for details)	X	
Edit and delete team discussions in all teams (for more information, see " Managing disruptive comments ")	X	

– Examples of access permissions for organization's owners and members

<https://help.github.com/enterprise/2.13/user/articles/permission-levels-for-an-organization/>

Github – Creating Organization



Secure | <https://github.sydney.edu.au/organizations/new>

Enterprise Search GitHub Pull requests Issues Explore

Sign up your team

✓ Completed
Create personal account

Step 2:
Create organization

Step 3:
Add members

Create an organization account

Organization name

✓

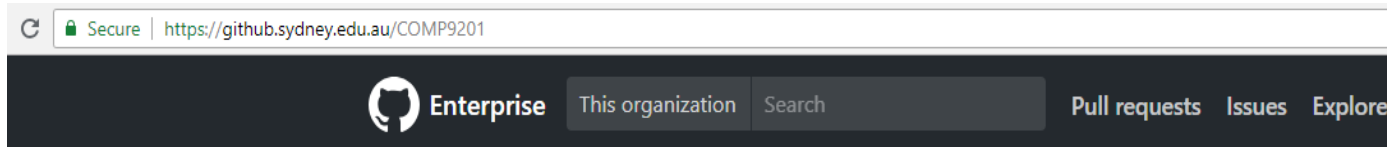
This will be your organization name on <https://github.sydney.edu.au/COMP9201>.

Contact email

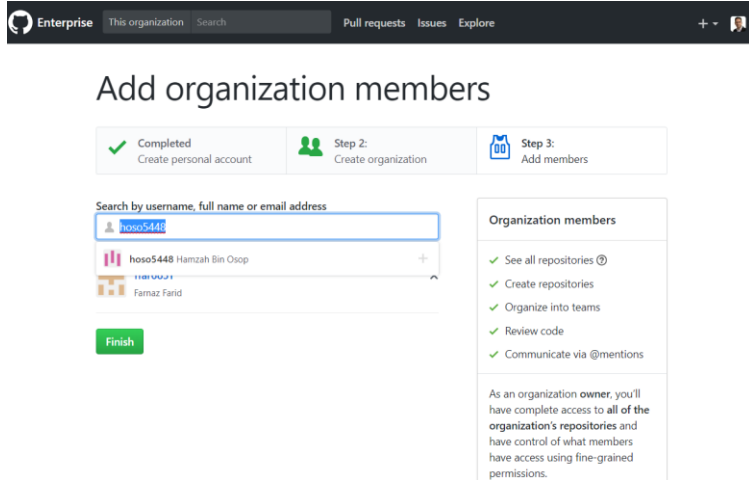
Create organization

Organization accounts allow your team to plan, build, review, and ship software — all while tracking bugs and discussing ideas.

Organizational accounts have a namespace where all their projects exist



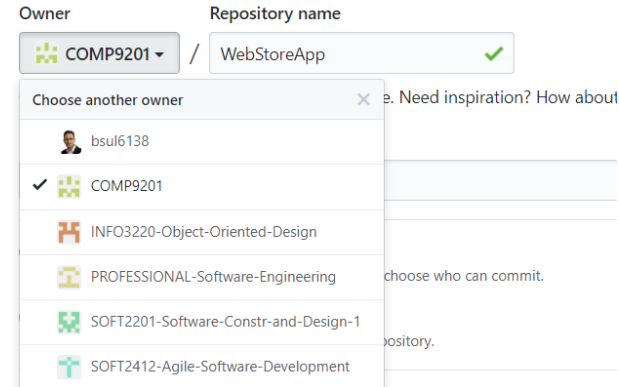
GitHub – Add Members to Organization



- Note: when you create a new repo you can create them under your personal account or under any of the organizations that you're owner in

Create a new repository

A repository contains all the files for your project, including the revision history.



GitHub Organization – Manage Repos

The screenshot shows the GitHub organization page for COMP9201. The top navigation bar includes 'Enterprise', 'This organization', 'Search', 'Pull requests', 'Issues', and 'Explore'. The main header displays the organization name 'COMP9201' and a 'Create a new repository' button. Below this, a message states 'This organization has no repositories.' The left sidebar contains tabs for 'Repositories', 'People', 'Teams', 'Projects', and 'Settings'. The 'Repositories' tab is active, showing a list of repositories: 'Front-end' (Private), 'Designs' (Private), 'Back-end' (Private), and 'WebStoreApp' (Private). The right sidebar shows the 'People' section with two members: 'bsul6138' (Basem Fathi Suleiman) and 'ffar6831' (Farnaz Farid).

The screenshot shows the GitHub repository page for COMP9201 / WebStoreApp. The top navigation bar includes 'Enterprise', 'This repository', 'Search', 'Pull requests', 'Issues', and 'Explore'. The main header displays the repository name 'COMP9201 / WebStoreApp' and a 'Private' label. Below this, a message states 'This repository has no repositories.' The left sidebar contains tabs for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Insights', and 'Settings'. The 'Settings' tab is active, showing the 'Teams' section. The 'Teams' section lists three teams: 'FrontEndDeve' (Front-end development team, 2 members, Read permission), 'BackEndDeve' (Back-end Development Team, 2 members, Admin permission), and 'Designers' (Web Application Designers, 1 member, Read permission). A 'Create new team' button is visible at the top right of the Teams section.

GitHub Organization – Manage People

COMP9201

Repositories 0 People 3 Teams 0 Projects 0 Settings

Find a member...

Members Outside collaborators Add member

Select all	2FA	Role	0 teams
Basem Fathi Suleiman bsul6138	2FA X	Private Owner	0 teams
Farnaz Farid ffar6831	2FA X	Private Member	0 teams
Hamzah Bin Osop hoso5448	2FA X	Private Member	0 teams

- Manage
- Change role...
- Convert to outside collaborator
- Remove from organization

Enterprise This organization Search Pull requests Issues Explore

COMP9201

Repositories 4 People 3 Teams 3 Projects 0 Settings

ffar6831 has access to 3 repositories

Find a repository they have access to...

Repository	Access Level	Manage access
COMP9201/WebStoreApp	Read on this repository	Manage access
COMP9201/Designs	Write on this repository	Manage access
COMP9201/Front-end	Admin on this repository	Manage access

Role: Member

3 repositories

1 team




Membership private

Two-factor security disabled

Convert to outside collaborator

Remove from organization

GitHub Organization – Manage Teams

Find a team...	Import teams	New team
Select all		
Visibility Members		
BackEndDeve Back-end Development Team	 2 members	0 teams
Designers Web Application Designers	 1 member	0 teams
FrontEndDeve Front-end development team	 2 members	0 teams

You may have 3 repos; Designs, Front-end and Back-end. You want FrontEndDeve to work on the Front-end and Designs repos, Designers team to work on Designs repo and BackEndDeve to work on Back-end repo

COMP9201 / BackEndDeve

Discussions

Members 2

Teams 0

Repositories 2

Find a repository...

Add repository

Select all		
COMP9201/Back-end	Private	Admin
updated 21 minutes ago		
COMP9201/WebStoreApp	Private	Admin
updated an hour ago		

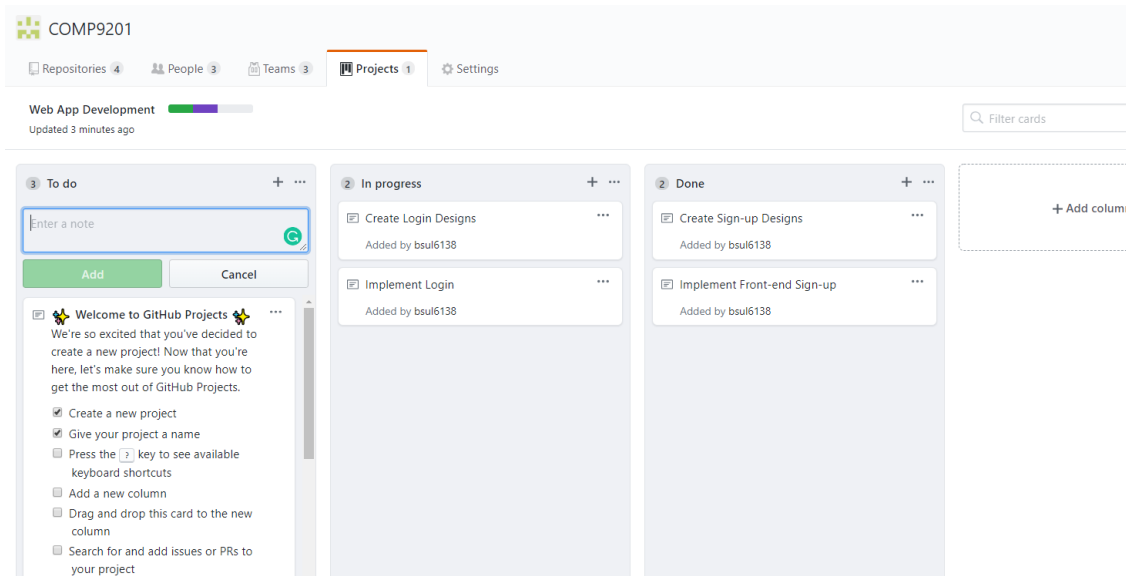
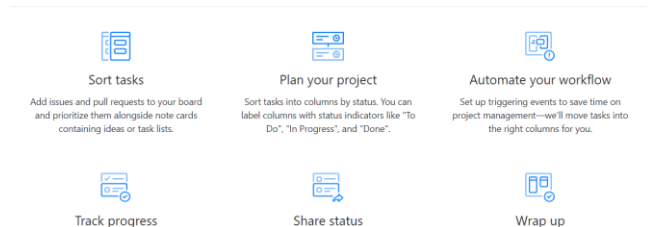
GitHub Organization – Manage Projects



Organize your issues with project boards

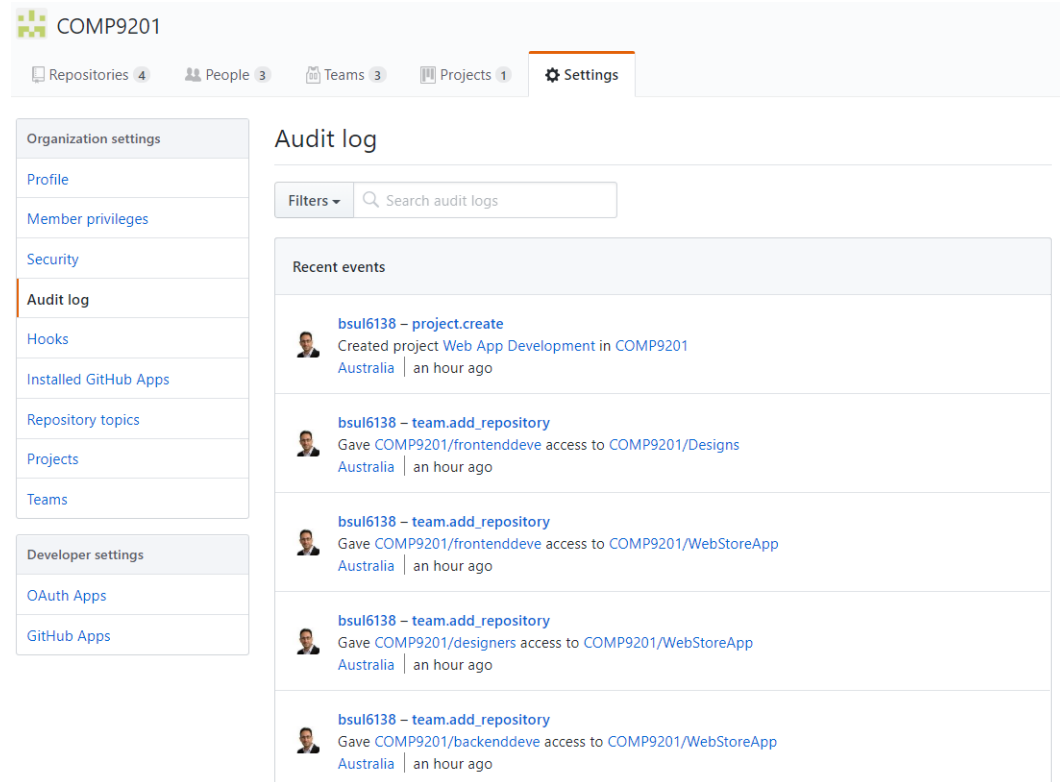
Did you know you can manage projects in the same place you keep your code? Set up a project board on GitHub to streamline and automate your workflow.

[Learn More](#) [Create a project](#)



GitHub Organization – Audit Log

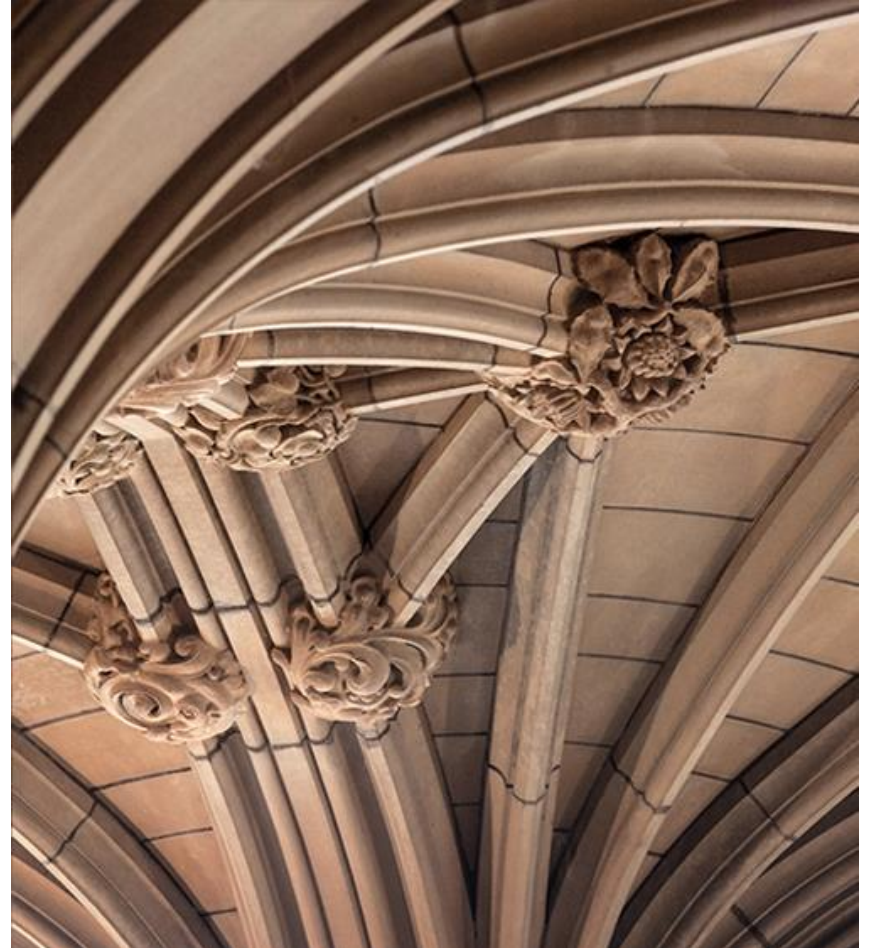
- Audit log records all events that have happened at the organization level, who did them and where in the world they were done



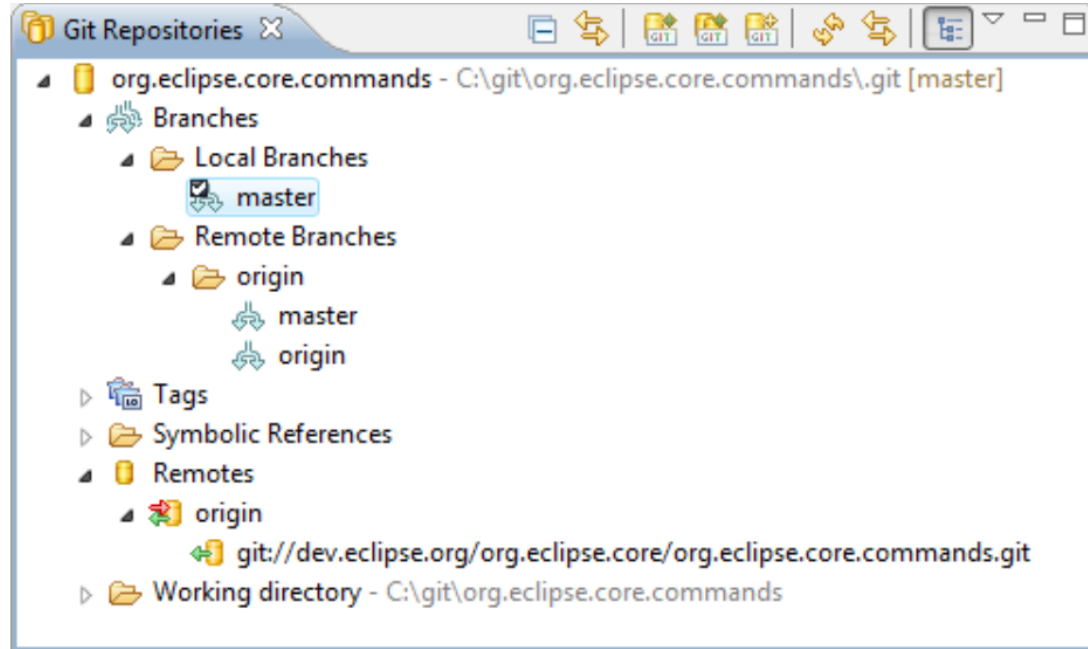
The screenshot shows the GitHub Organization Audit Log for the organization COMP9201. The interface includes a top navigation bar with tabs for Repositories (4), People (3), Teams (3), Projects (1), and Settings. A left sidebar contains a menu for Organization settings (Profile, Member privileges, Security, Audit log, Hooks, Installed GitHub Apps, Repository topics, Projects, Teams) and Developer settings (OAuth Apps, GitHub Apps). The main content area is titled 'Audit log' and features a search bar and a 'Recent events' section. The events listed are:

- bsul6138 – project.create**
Created project [Web App Development](#) in COMP9201
Australia | an hour ago
- bsul6138 – team.add_repository**
Gave COMP9201/frontenddeve access to COMP9201/Designs
Australia | an hour ago
- bsul6138 – team.add_repository**
Gave COMP9201/frontenddeve access to COMP9201/WebStoreApp
Australia | an hour ago
- bsul6138 – team.add_repository**
Gave COMP9201/designers access to COMP9201/WebStoreApp
Australia | an hour ago
- bsul6138 – team.add_repository**
Gave COMP9201/backenddeve access to COMP9201/WebStoreApp
Australia | an hour ago

Git in Development Environments

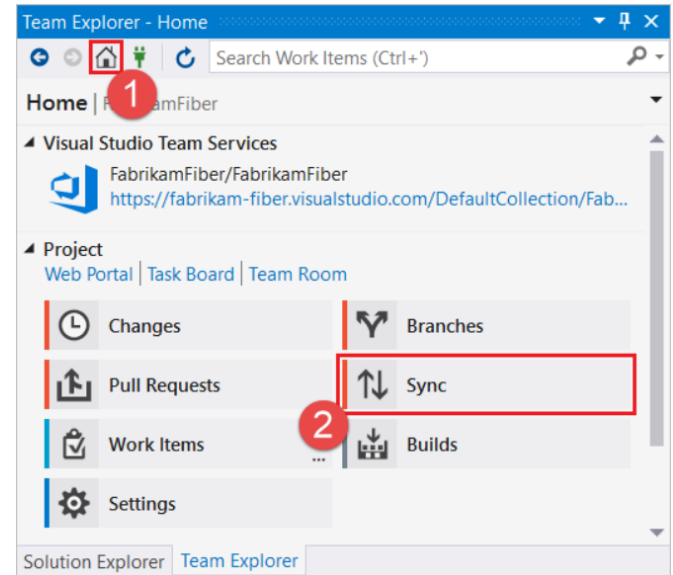
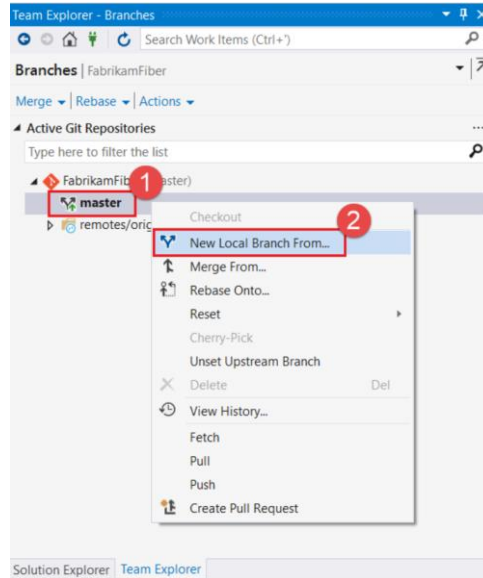
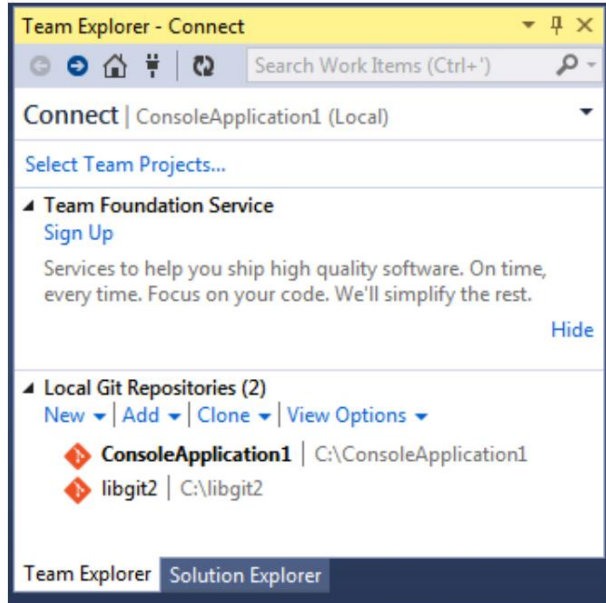


Eclipse Plugin – Egit



<https://www.eclipse.org/egit/>

Git in Visual Studio



<https://docs.microsoft.com/en-us/vsts/repos/git/gitquickstart?view=vsts&tabs=visual-studio>

Tutorial/Lab. work

Collaborating on a remote repo using GitHub

GitHub



References

- Scott Chacon. 2014. Pro Git (2nd ed.) Apress
 - Free online book – download from <https://git-scm.com/book/en/v2>
- Additional Resources – Paper
 - H-Christian Estler, Martin Nordio, Carlo A. Furia and Bertrand Meyer: *Awareness and Merge Conflicts in Distributed Software Development*, in proceedings of ICGSE 2014, 9th International Conference on Global Software Engineering, Shanghai, 18-21 August 2014, IEEE Computer Society Press (best paper award),
 - http://se.ethz.ch/~meyer/publications/empirical/awareness_icgse14.pdf