Open Source SW & Lab - Summer 2023

5. Distributed Git

Walid Abdullah Al
Computer and Electronic Systems Engineering
Hankuk University of Foreign Studies

Based on:
Pro Git (2022) by Scott Chacon, Ben Straub

With a remote Git

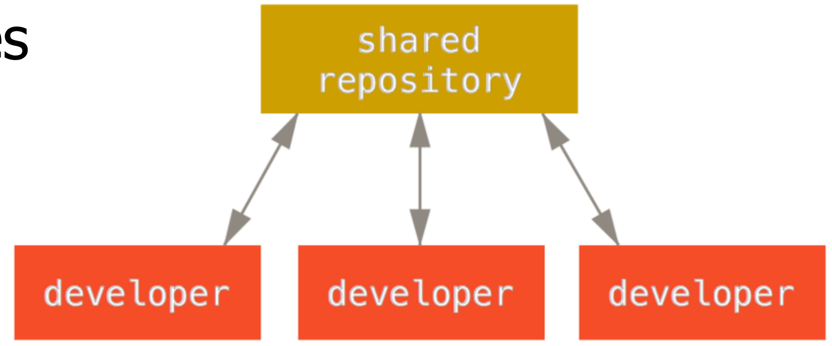
- **What you could do now**
  - Have a remote repository
  - Clone/fetch/pull that repository into your local
  - Work in your local
  - Push changes to remote
- **But how to collaborate?**

Distributed workflows

- **In Git,**
  - Every developer is a node and a hub
  - Can contribute to other’s public repository
  - Can have their own public repository that others can contribute to
- **Three major workflows**
  - Centralized workflow
  - Integration-manager workflow
  - Dictator and Lieutenants Workflow

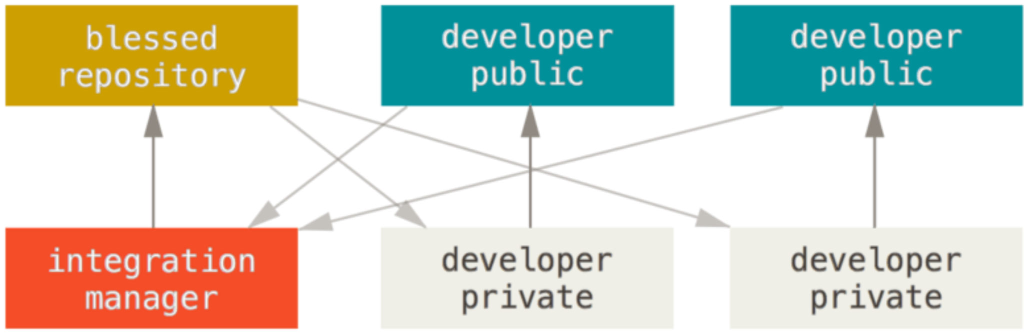
Centralized workflow

- **One central hub**
  - Public repository that can accept codes/contributions
- **Multiple nodes**
  - Developers/contributors with equal access
  - Who must synchronize their works with the hub
- **Push problem:** only fast-forward changes are allowed
  - Two developers cloned the same repo
  - Both made changes
  - 1st developer pushes changes
  - 2nd developer can't push
    - He must merge the changes of the 1st developer then push



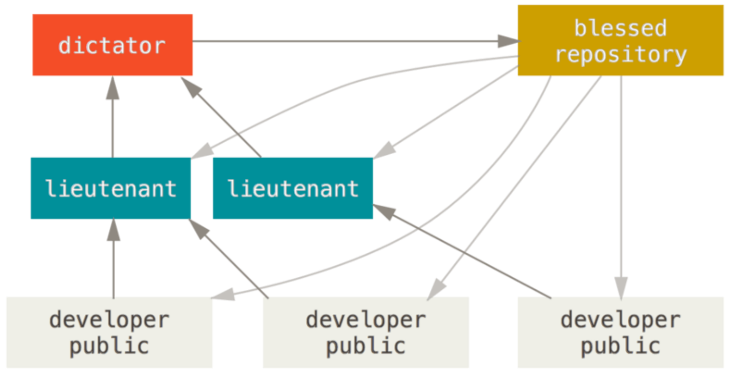
Integration-manager workflow

- A **contributor**
  - clones the main repository and makes changes
  - pushes to their **own public copy**
  - Send **pull request** to maintainer
- A **maintainer**
  - pushes merged changes to the main repository.



Dictator and lieutenants workflow

- **Developers**
  - work on their topic branch and rebase their work on top of master
- **Lieutenants**
  - merge the developers' topic branches into their master
- **Dictator**
  - merges the lieutenants' master branches into the dictator’s master
  - pushes that master branch to the reference repository so the other developers can rebase on it



Contribution: private small team

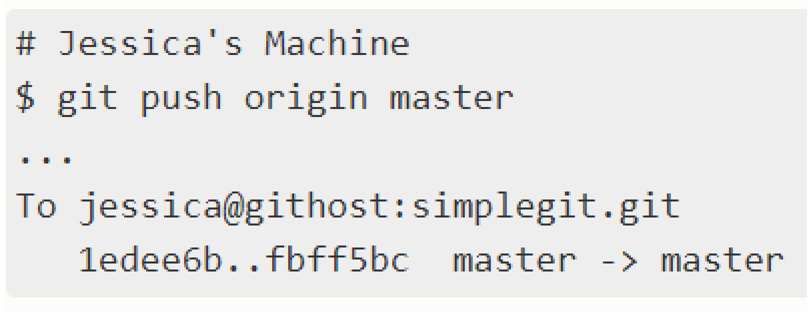
- **Each developer has equal access**
- **Say:**
  - John and Jessica started collaborating on a project
  - (the cloning step)

```
# John's Machine
$ git clone john@githost: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(-)
```

```
# Jessica's Machine
$ git clone jessica@githost: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(-)
```

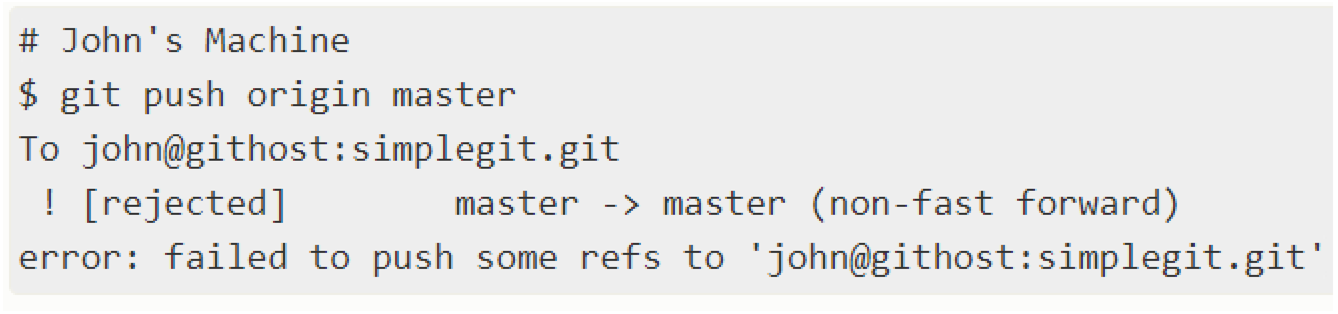
Contribution: private small team

- **Jessica pushes first without any problem**



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

- **John tries to push but failed**
    - (not fast-forward changes)

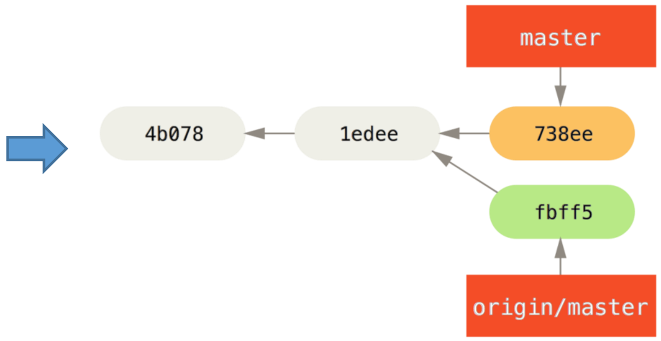


```
# John's Machine
$ git push origin master
To john@githost:simplegit.git
 ! [rejected]        master -> master (non-fast forward)
error: failed to push some refs to 'john@githost:simplegit.git'
```

Contribution: private small team

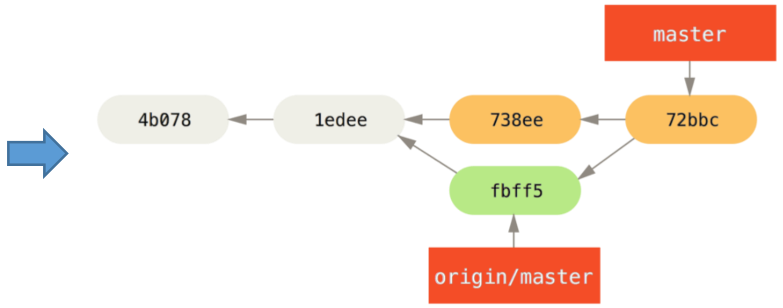
- **How can john push his branch?**
  - (1) Fetch the updated origin

```
$ git fetch origin
...
From john@githost:simplegit
 + 049d078...fbff5bc master     -> origin/master
```



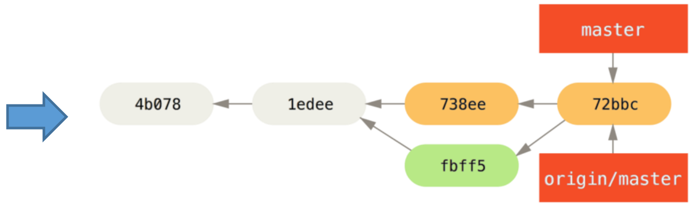
  - (2) merge into local master

```
$ git merge origin/master
Merge made by the 'recursive' strategy.
 TODO |     1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
```



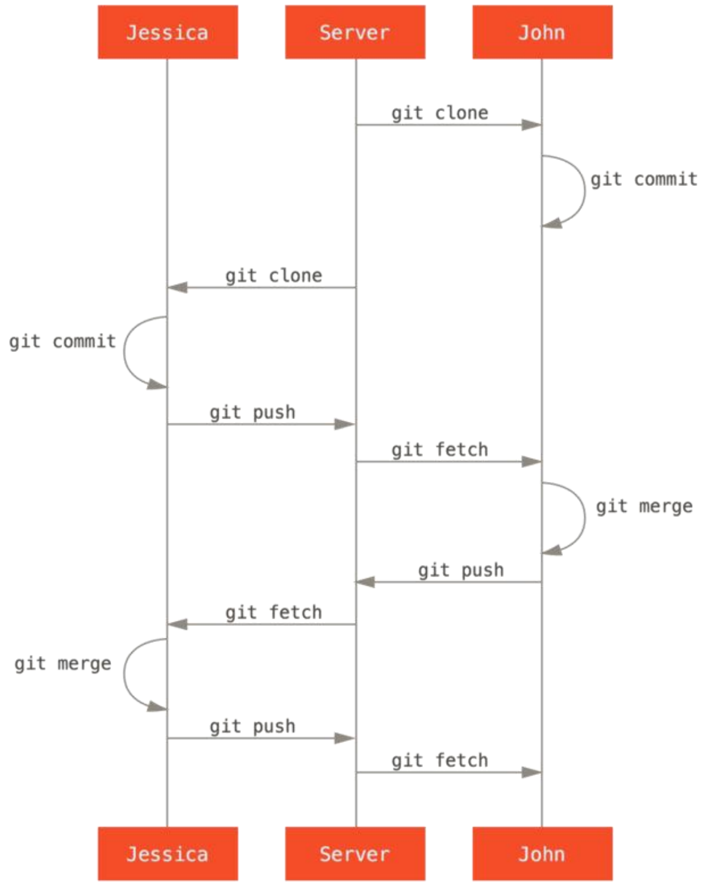
  - (3) push

```
$ git push origin master
...
To john@githost:simplegit.git
   fbff5bc..72bbc59  master -> master
```



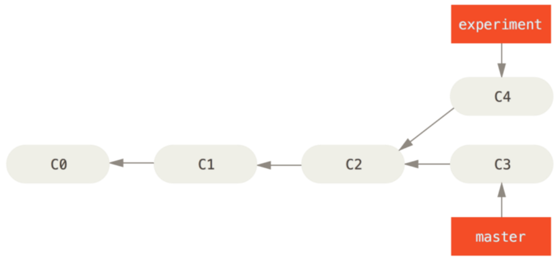
Contribution: private small team

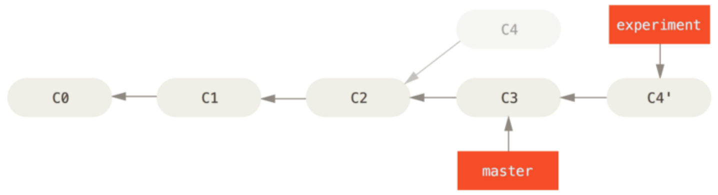
- **A simple multi-developer workflow**



Rebasing vs Merging (1)

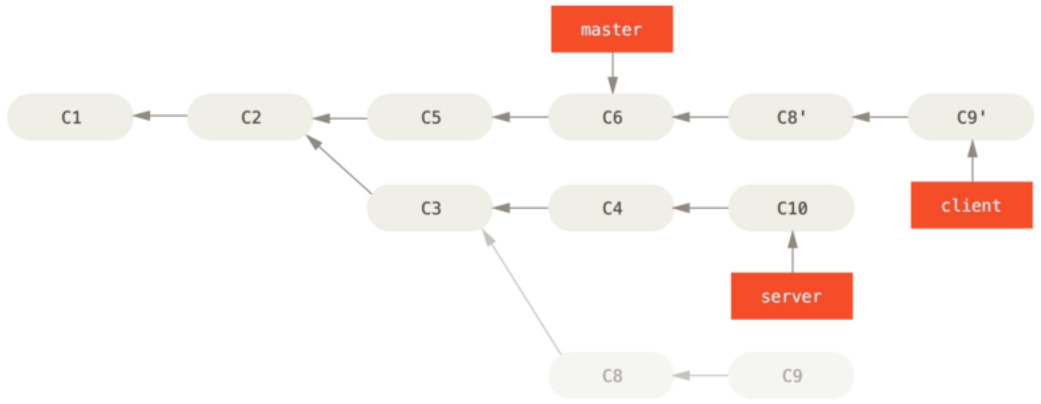
- **Merging**
  - Merge two divergent branches
- **Rebasing**
  - Integrate changes from one branch into another





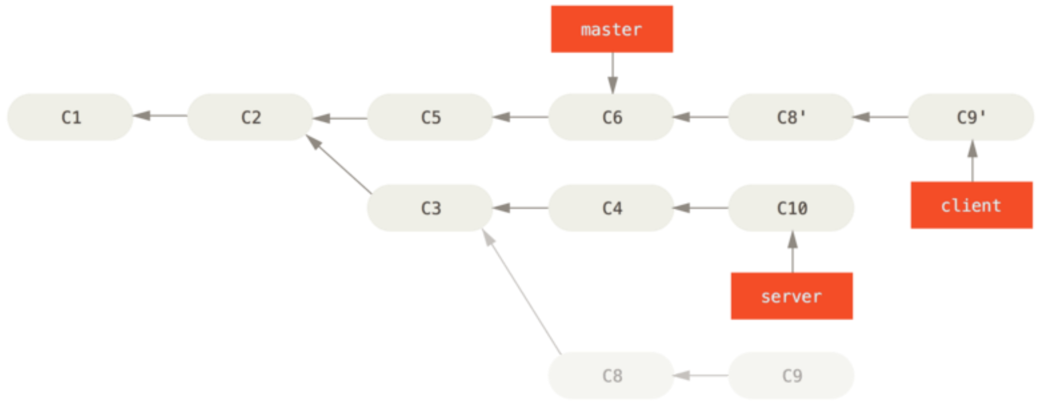
Rebasing vs Merging (2)

- **Merging**
  - All commits are combined into a single commit
- **Rebasing**
  - All commits are rebased
  - Same number of commits added



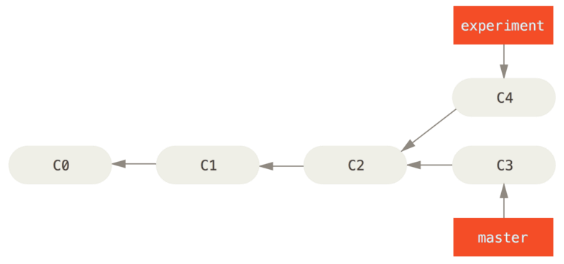
Rebasing vs Merging (3)

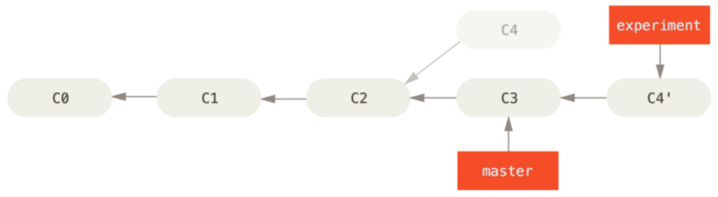
- **Merging**
  - Best used target branch is shared
- **Rebasing**
  - Best used when target branch is private



Rebasing vs Merging (4)

- **Merging**
  - Preserves history but complex log
- **Rebasing**
  - Simpler log but alters/rewrites history





Team Assignment

- Step-1: Select your team on Eclass
  - Required for upcoming labworks and project
  - If you do not have a preferred team, I will randomly assign

Team Assignment (contd.)

- Step-2: add your info to our Github repository
  - Task-1: accept the invitation to **join OSS-2023**
  - Task-2: add the following remote rep.: https://github.com/OSS-2023/teams
    - Fetch and merge this rep. to your local git to start working
  - Task-3: add your name and github id in your team file (teamX.md) and push
    - Member1, member2, member3 should be based on the Eclass team sequence.

Team Assignment (contd.)

- Step-3: Team collaboration on a small project
  - Now make a copy of codes/complex_sample.py
  - Rename it as: complex_X.py
    - X: your team number (e.g., complex_1.py)
  - Complete the function assigned to you
  - And push

Done for today!

- Please, ask me to check your attendance