Version Control Systems: Part 1

Working with offline repositories, sharing repositories and cloning repositories.

VERSION CONTROL SYSTEMS

Version control systems are an invaluable tool when developing software, providing an effective way of tracking the changes you make to a piece of source code as you develop a piece of software.

In addition, version control systems manage source code, giving you, and other developers, the ability to edit a single project, or even a single file in that project, concurrently and in a predictable manner.

http://www.tratt.net/laurie/teaching/2015_2016/apt/session3.pdf

https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

WHY SHOULD I CARE? (1)

1. If you want a job as a software developer you must be able to demonstrate experience with version control systems.

2. Working collaboratively on code, as you will be doing in your PPA major piece of coursework, becomes extremely inefficient without version control.

3. Accessing code, for example in the PPA lectures, can be expedited by having good version control integration with an IDE.


if you use dropbox for version control
INSTRUCTOR
you're gonna have a bad time
quickmeme.com

WHY SHOULD I CARE? (2)

4. Other services (e.g. Dropbox) are sub-optimal when it comes to maintaining versions of your code and supporting collaborative development.

5. You will be unable to start your group project without using and establishing a version control system for your group.

6. A portion of the marks for the PPA major piece of coursework are based upon the establishment of, and on your activity using, a version control system.

GIT: THE NOT-SO-STUPID CONTENT TRACKER

Git is a modern version control system, and a very powerful one at that. We will likely on be using a subset of its available features, but feel free to explore its more advanced features, if you are confident and curious.

We think about git in terms of repositories, which are either held locally (for your own personal version control), or hosted using an online service (for distributed version control):

REPOSITORIES

A git repository is just a directory containing content, like any other, except it is managed by git (the version control system), which will, predominantly, maintain versions of the content in the repository.

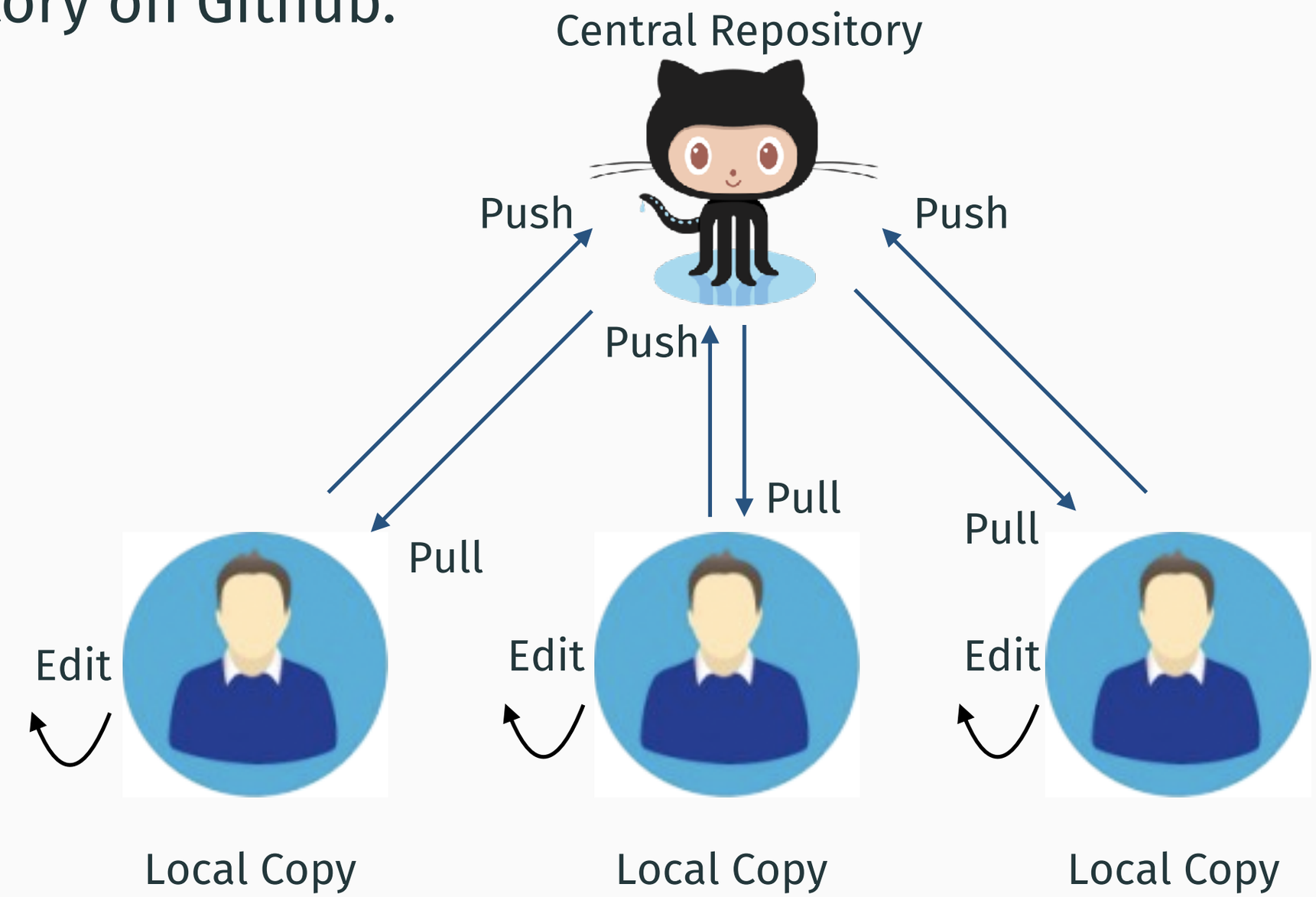
- Also refers to a utility folder (.git) used by the version control system.

You don’t necessarily have to store code inside a git repository, you can also used it to store versions of most files, for example written assignments.

For the PPA Major Piece of coursework, you will create a central git repository for your group on a repository hosting service called Github (King’s Enterprise Version).

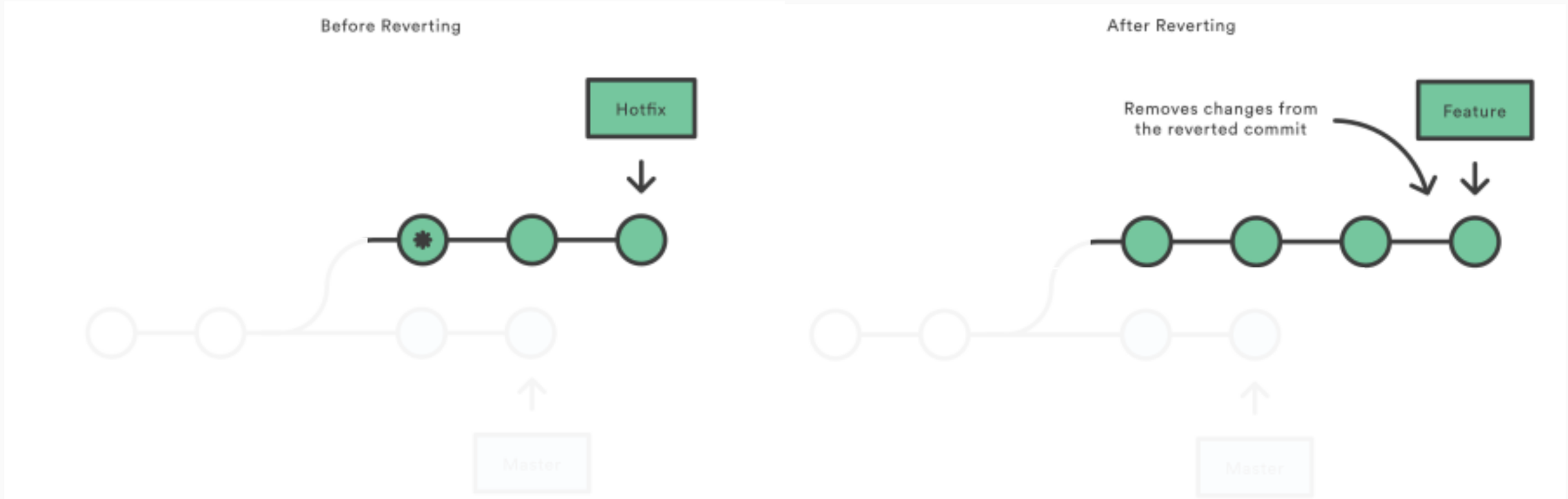
CENTRAL REPOSITORY OVERVIEW

All group members will then be able to keep a local copy of your collective solution, which is synchronised with the central repository on Github.



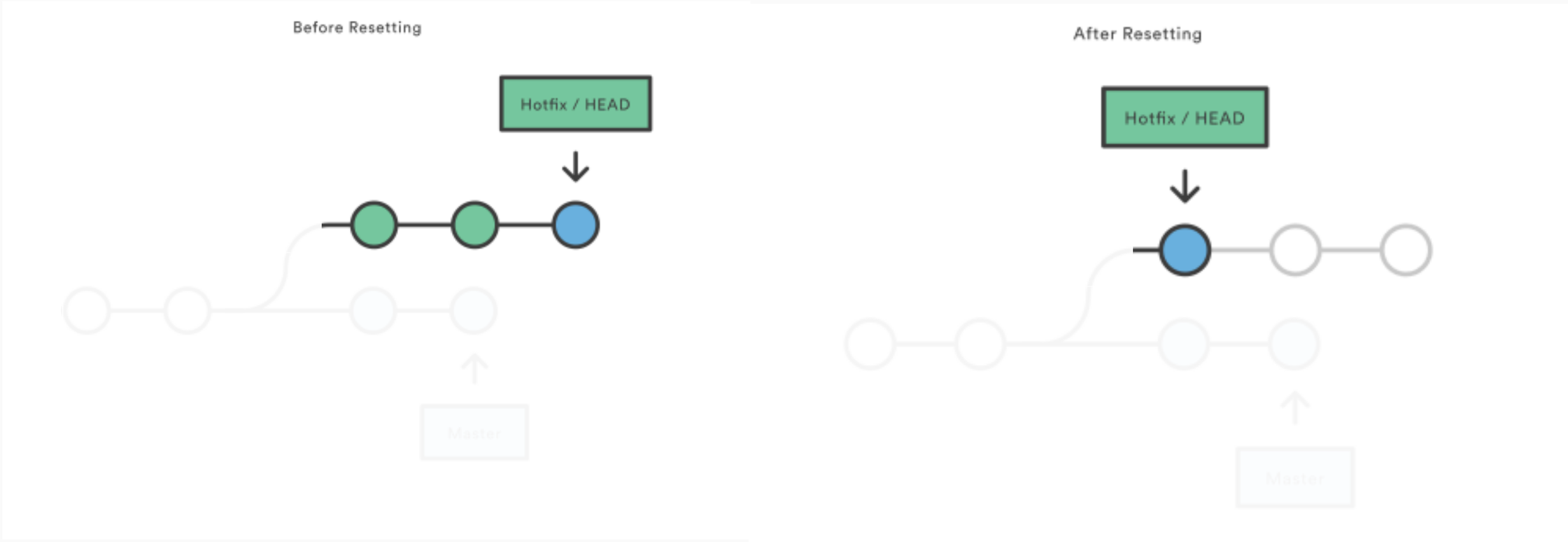
Git, in addition to maintaining versions, manages this synchronisation, and facilitates simultaneous editing.

REVERTING



https://www.atlassian.com/git/tutorials/resetting-checking-out-and-reverting

RESETTING



https://www.atlassian.com/git/tutorials/resetting-checking-out-and-reverting

Version Control Systems: Part 2

CWX Requirement 0, Version Control for the Major Piece of Coursework.

If you have not yet been allocated to a group, or you do not yet have four members in your group, you must wait until you have been allocated to a group, or you have received additional members, before embarking on any of the steps listed from here onwards.

This will avoid you having to repeat various setup steps, as well as ensuring you avoid other other issues with the setup process, which may affect your marks.

Just watching through this video is, of course, fine.

MAJOR COURSEWORK (CWX), REQUIREMENT 0 (5 MARKS) (1)

The video details a series of tasks, each of which is to be completed by **either everyone in your group**, or **one person in your group**.

- If you cannot decide who this single person is, then **nominate** the person who **started your team on TeamFeedback**.

There is **usually** a mark attributed to each task.

This will be a **collective mark** awarded to **everyone in the group**, and represents a portion of the 5 marks available for Requirement 0 in the Major Coursework (CWX) assignment.

MAJOR COURSEWORK (CWX), REQUIREMENT 0 (5 MARKS) (2)

Some tasks do **not** carry a mark, as they exist to support **later** tasks in the video, which do carry a mark.

- These are mainly the tasks detailed in the **first video**, which you may need to view again.

The deadline to complete these tasks is **Friday 10th February** at **7.00pm** (marks will not be awarded after this time), at which point the specification for the **remaining** CWX tasks will be released.

- There is no formal submission of these tasks, but they must be **visible** by this time, in order to receive the available marks.

MAJOR COURSEWORK (CWX), REQUIREMENT 0 (5 MARKS) (3)

We recommend **meeting with your group as soon as possible**, or at **least communicating over email**, to **discuss how you will carry out these steps**.

Although only worth 5 marks, **you can only receive the specification for the remaining major coursework marks (95 marks) after completing these tasks** (even after its official release date). So it is in your interest to complete them, and to do so in good time. More details later.

MAJOR COURSEWORK (CWX), REQUIREMENT 0 (5 MARKS) (4)

Already done:

1. Downloading and installing the Git application (0 marks) **To be completed by every group member.**

2. Checking GHE Credentials (0 marks) **To be completed by every group member.**

Summary of the tasks in the remainder of the video:

3. Creating your central repository (1 mark) **To be completed by one group member.**

4. Creating the project structure (1 mark) **To be completed by one group member.**

5. Creating a development branch, editing the branch, and merging with the master branch (3 marks) **To be completed by the remaining group members (and the one group member)**.

Task 1: Downloading and installing Eclipse (containing Git) or the standalone Git application.

Task 2: Checking GHE credentials

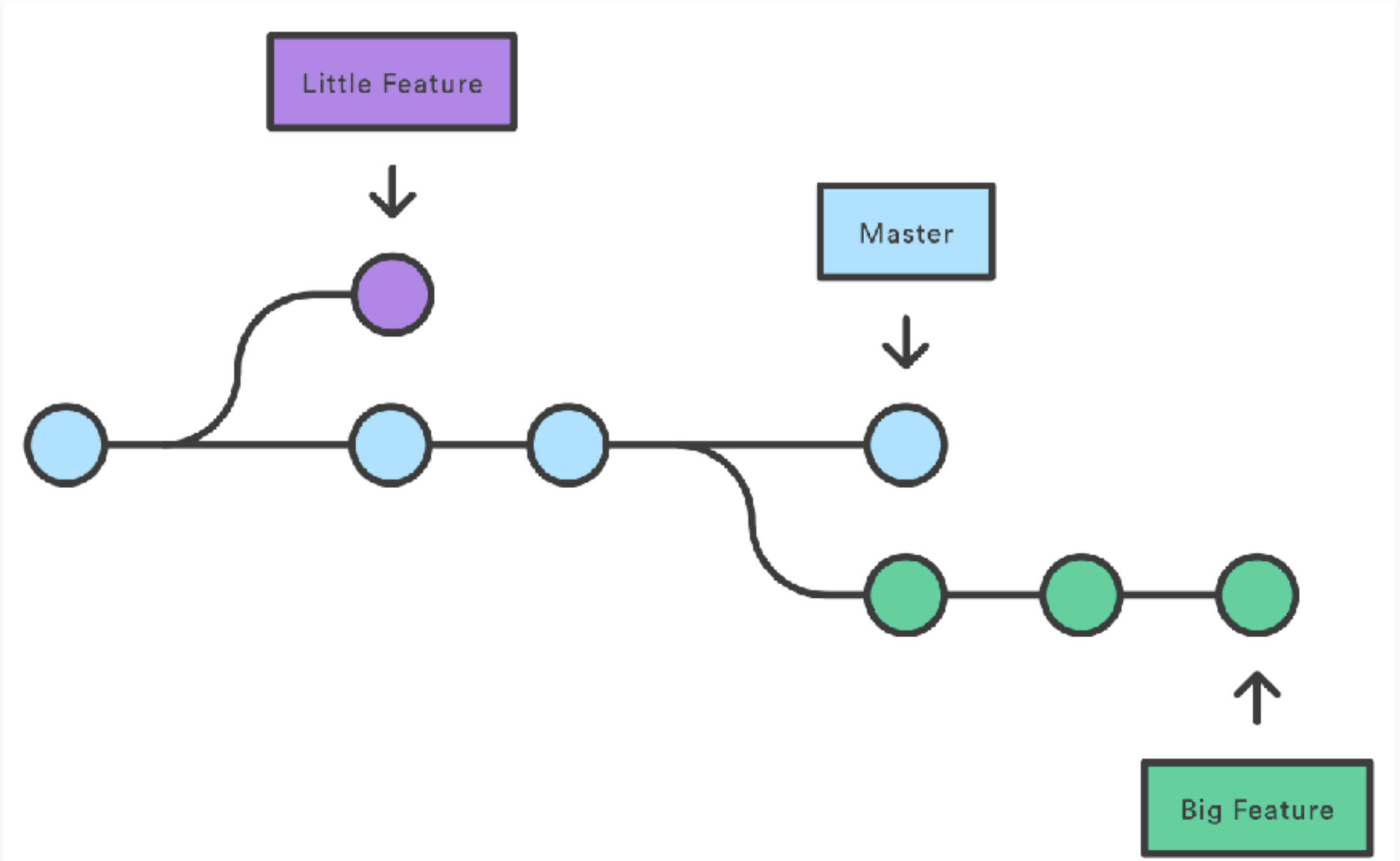
Task 3: Creating a central repository

The repository you create on King's Github Enterprise for your major piece of coursework must be private.

Task 4: Creating the project structure

Do not commit to the master branch directly.

BRANCHES

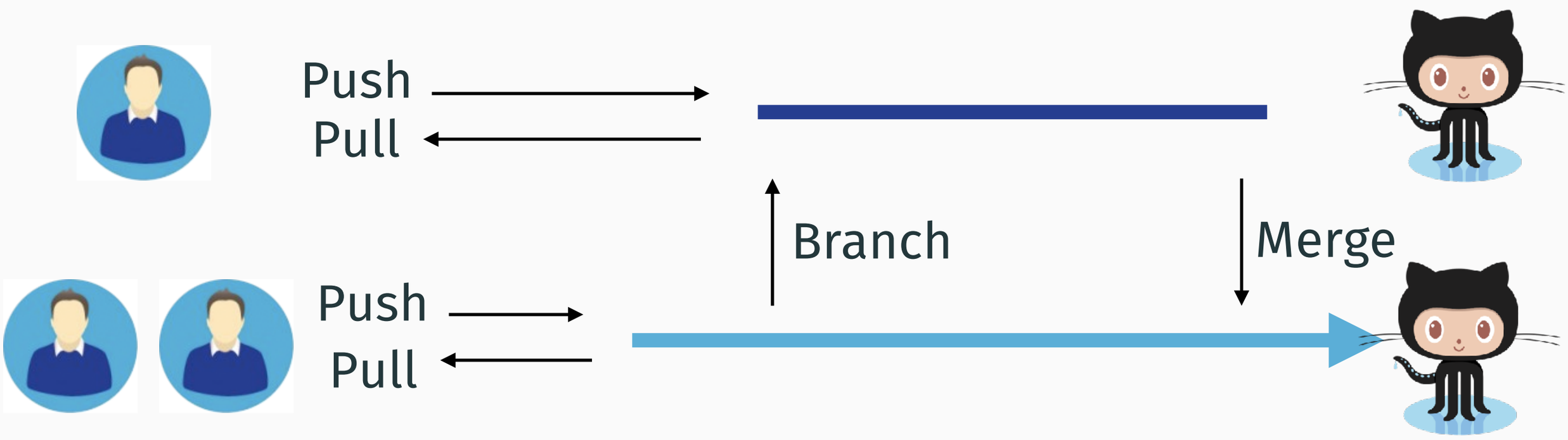


https://www.atlassian.com/git/tutorials/using-branches

BRANCHING OVERVIEW (1)

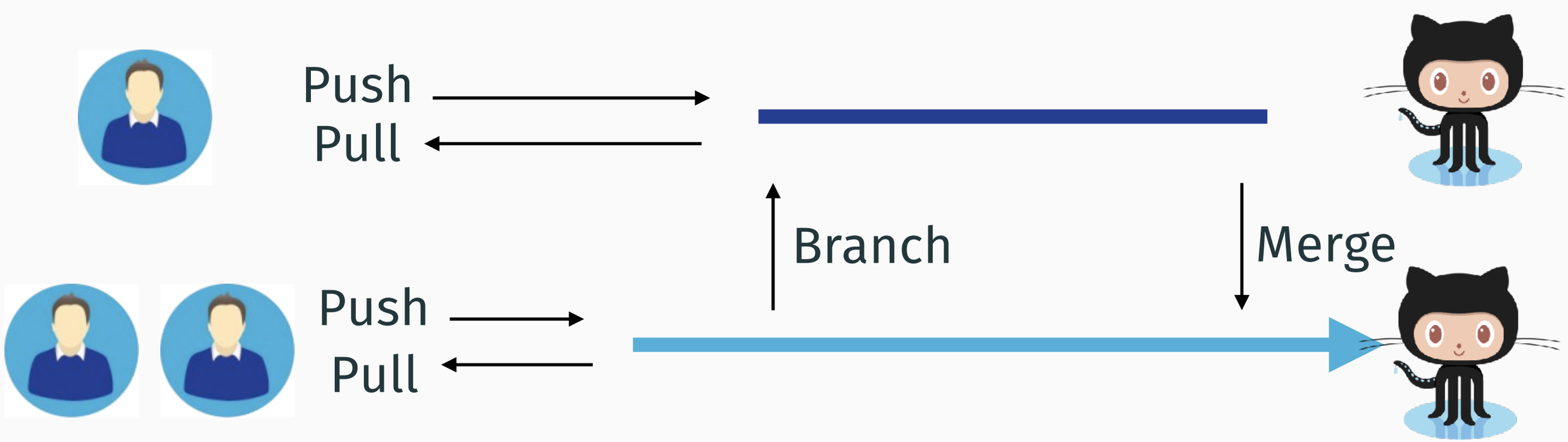
Branching assigns a **self-contained copy** of the central repository to a particular developer (typically), which can be versioned, committed to and even pushed **without affecting** the central repository.

This helps to identify **conflicts** between the edits made by different developers, without **losing** any work.



BRANCHING OVERVIEW (2)

Eventually the changes in the branch should be added to the master branch (i.e. shared with the other developers). This is known as **branch merging**, and may also require the resolution of **merge conflicts**, although good team communication should negate this possibility.



Task 5: Creating a development branch, editing the branch, and merging with the master branch (3 marks)

Git and Github are not a replacement for proper team management.

FEATURE BRANCHES

Just because you must be careful when merging one of your feature branches with the master branch, does not mean that multiple people can never work on the code in a repository at the **same time**.

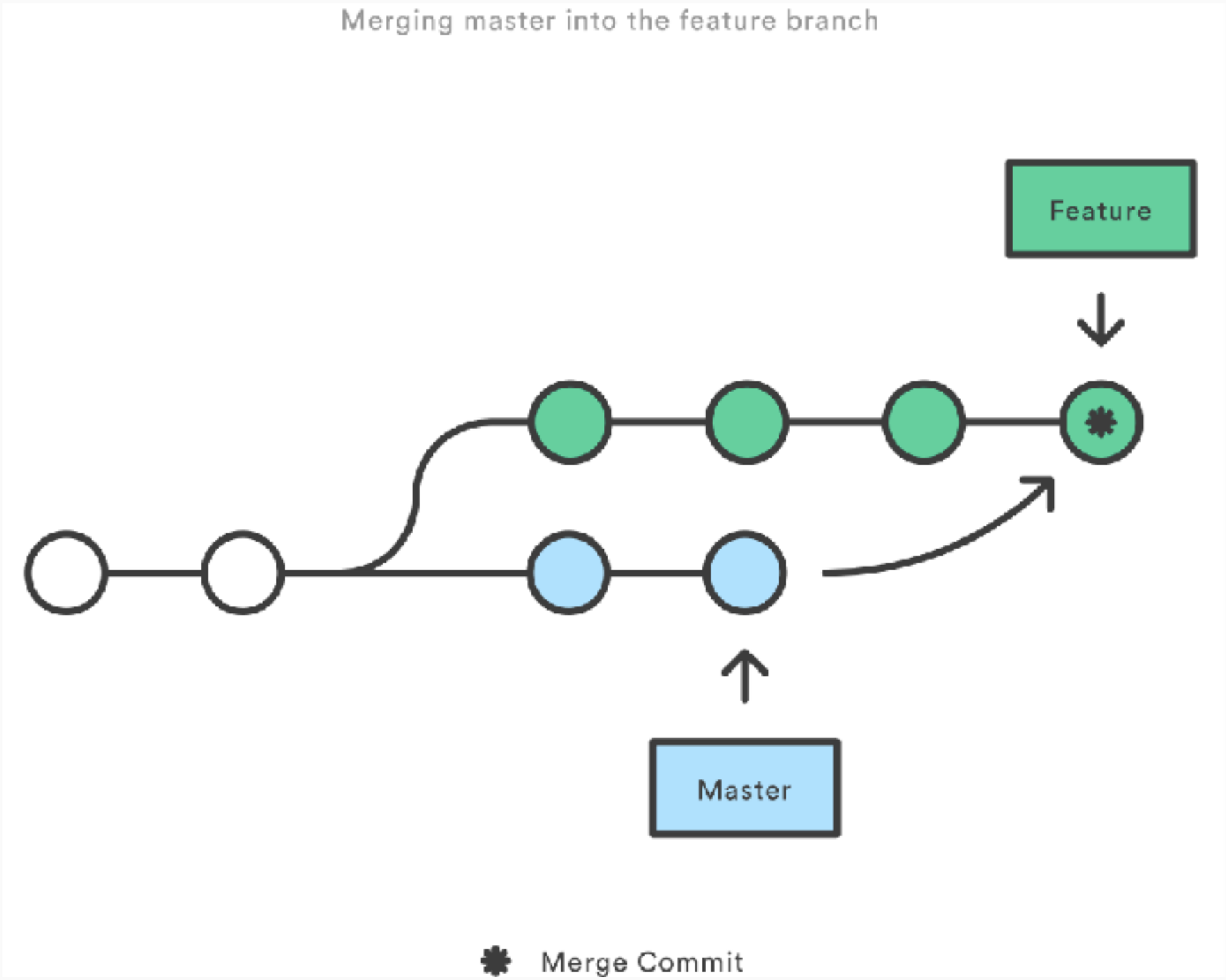
Typically, several branches will be created and used to work on **different parts** of a program, such that conflicts are unlikely to occur.

So, while care should be taken when merging, you should still strive to develop features in **parallel**.

Pull, from the central repository,
before making any changes.

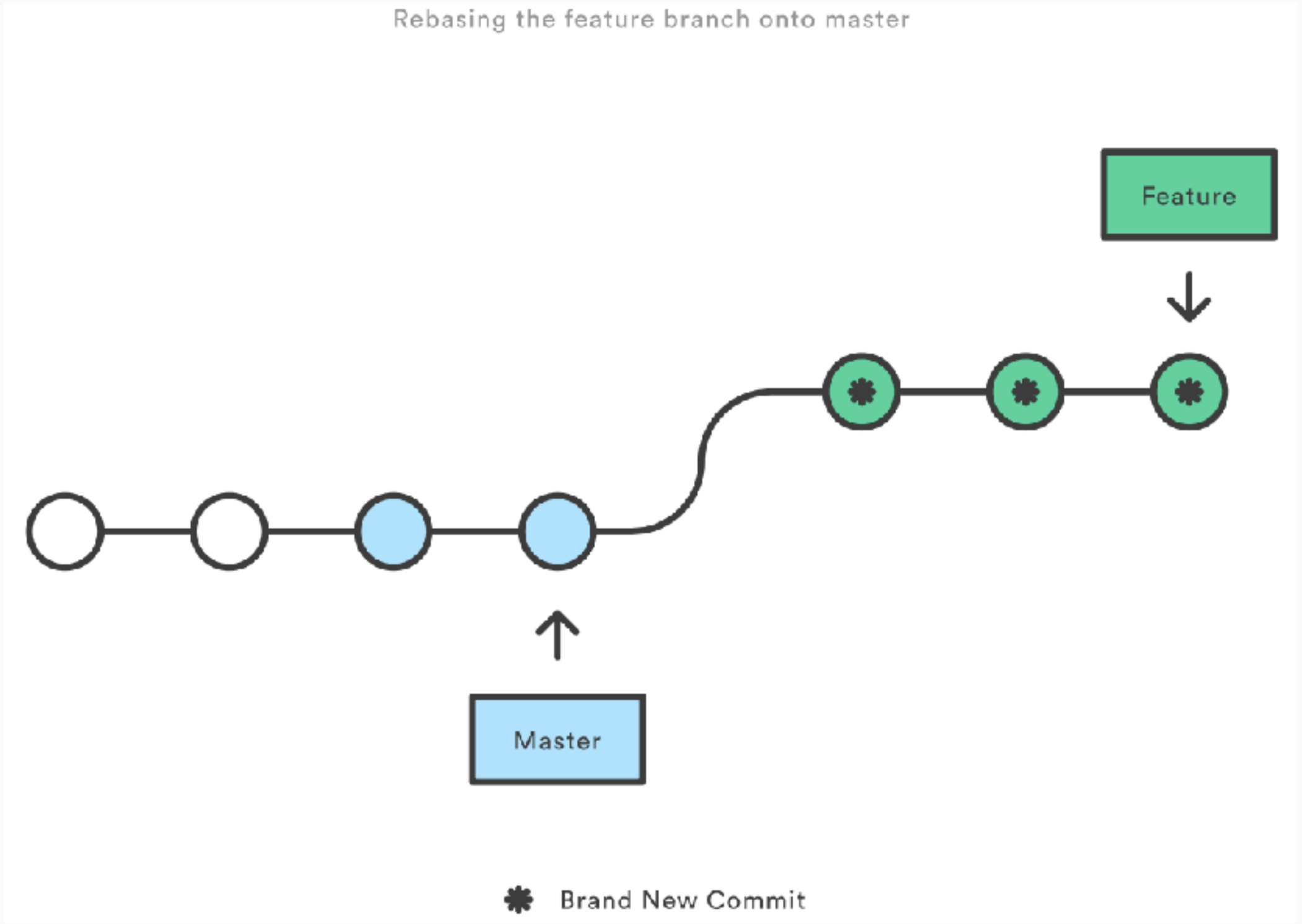
In fact, pull before doing anything.

MERGING INTO A FEATURE BRANCH FROM THE MASTER BRANCH



https://www.atlassian.com/git/tutorials/merging-vs-rebasing

REBASING



https://www.atlassian.com/git/tutorials/merging-vs-rebasing

If things go really wrong in your cloned copy of a repository, and there is a stable version in the central repository, simply delete and re-clone.

THE FINAL STATE OF YOUR CONTRIBUTORS FILE

```
public class Contributors {

   public static void main(String[] args) {

      System.out.println("Repository owner's name");
      System.out.println("Second group member");
      System.out.println("Third group member");
      System.out.println("Fourth group member");

   }

}
```

Double check that your k-number (or another reasonable identifier) is shown next to your commits on GHE, so we can attribute your commits back to you.

SUMMARY: INDIVIDUAL GIT WORKFLOW

Summary of Git use in this task

Central Repository

`Synchronise the local copy with the central repository'

Push

Local Copy

Add

Commit

`Tell the local copy about files'

`Save a version of those files'

Edit

SUMMARY: BRANCHES

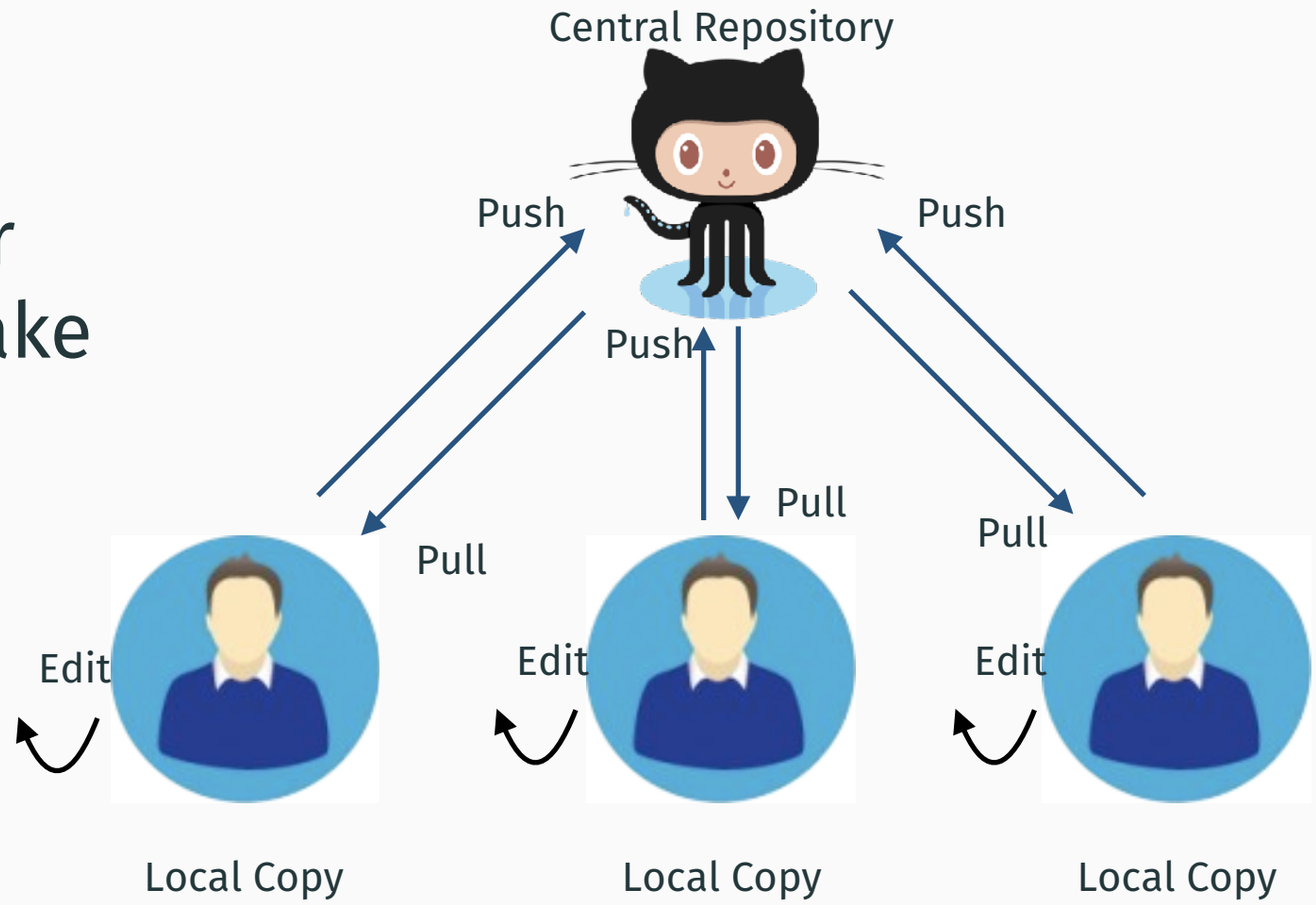
Branches differ from direct clones with a single master branch because the expectation is that a new version of a cloned repository is only committed (and subsequently pushed) if the author, and all the other developers are **happy** for the version to be a part of the `main' repository (the master branch).

In contrast, branches are **self-contained copies** that can be versioned separately, committed to, and worked on, and then later added back into the main repository.

This process of adding a branch back to a repository is called **branch merging**.

SUMMARY: COLLECTIVE GIT WORKFLOW

The workflow shown in the diagram earlier should now make more sense.

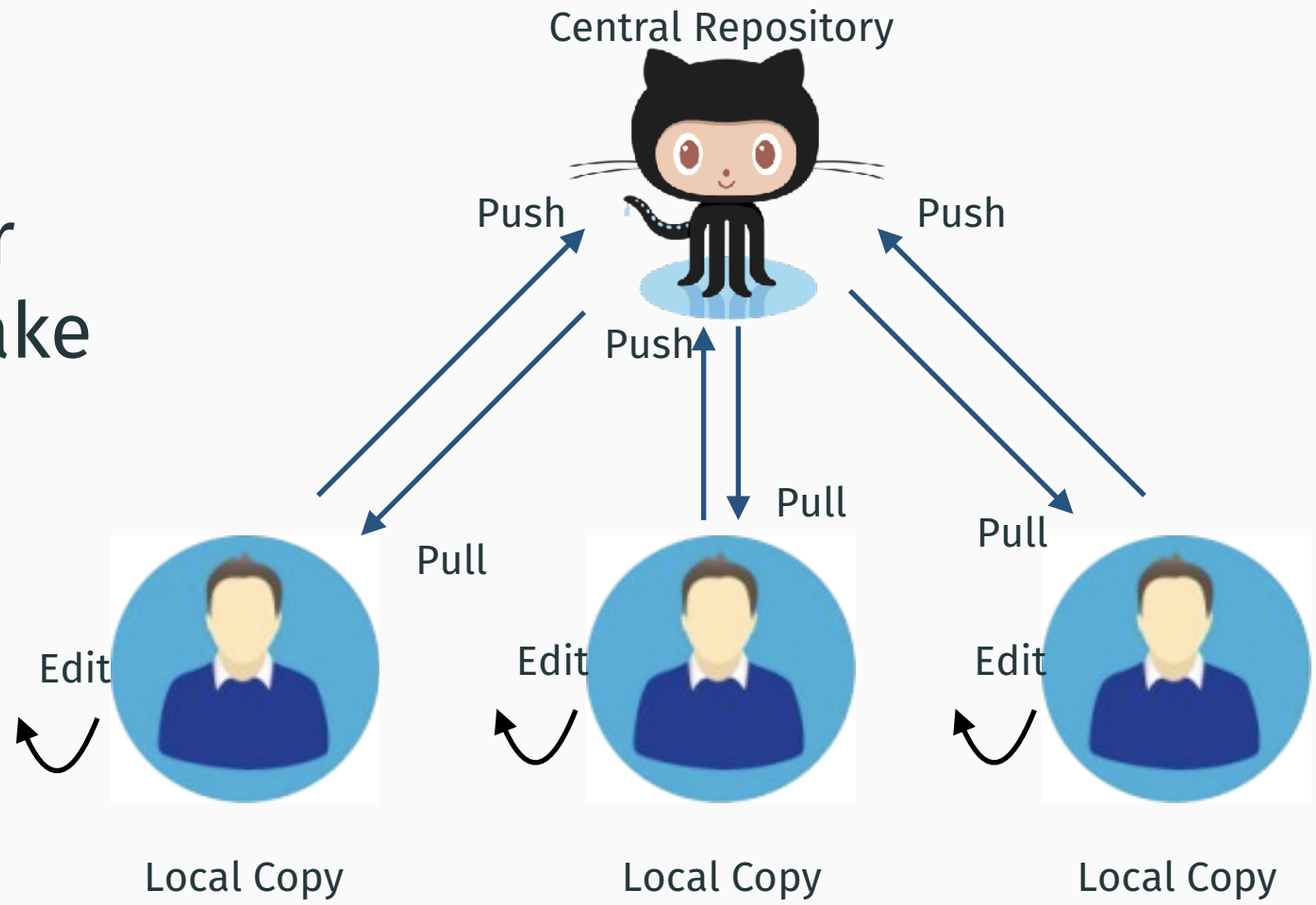


Each team member **clones**, then **pulls**, **edits** (then **commits**) and then **pushes** (synchronises, or merges, with the central repository).

When working on the same source file (i.e. editing and merging), like you are doing in this task, **it is strongly recommended that you agree on an order in which to edit and push**, in order to avoid merge conflicts.

SUMMARY: COLLECTIVE GIT WORKFLOW

The workflow shown in the diagram earlier should now make more sense.



Merge conflicts occur when two people edit the same line(s) of a file differently and then push to the same central repository.

This issue cannot be negated automatically (only by careful team organisation, or experimental merge techniques), but it can certainly be better managed by using **branches**. Branching also has a number of other benefits.

Do not delete the initial branches you have created for Requirement 0 from your repository.

Other branches you create can be deleted as you wish.

REFERENCES AND FURTHER READING

https://git-scm.com/documentation

http://www.tratt.net/laurie/teaching/2015_2016/apt/session3.pdf

are the most useful. There is also the **first video** on version control, available on KEATS, and as much material as you could ever want online.

As always, the `Your Questions / Our Answers' **forum** on KEATS is a good place to ask version control queries.

It will likely take a fair bit of **experimentation** before everything in this video makes sense. **The best way to learn version control is to try it out.**

SUMMARY

Only by completing the tasks shown in this video will you be able to receive a copy of your major coursework assignment.

In addition, these tasks are worth 5 marks (5%) towards the major coursework grade of everyone in your group. Therefore, although most tasks must be carried out individually, their completion will impact the grade of **everyone in your group**.

Look carefully at which tasks need to be carried out by **one** group member, and which tasks need to be carried about by **all** (or the remaining) group members.

Reminder: The deadline to complete these tasks (i.e. have them visible in a repository where `MC' is a contributor) is **Friday 10th February at 7pm**.

IMPORTANT: PROVING YOUR INVOLVEMENT IN THE MAJOR PROJECT

With your repository set up, you and your group members should now make **regular commits** on your branches, **carefully** merge these commits, and then **push them**.

By making regular commits, you **prove to us that you were involved in the construction of the solution you submit at the end of the term**.

Students without sufficient commits, and poor feedback from their group members, risk having their final project mark scaled down to reflect their contribution.