# CM-2603 Data Science Group Project

## Introduction To Version Control Systems

Week 10 | Prasan Yapa | Sriyan Fernando

ROBERT GORDON UNIVERSITY ABERDEEN
TEF Gold
INFORMATICS INSTITUTE OF TECHNOLOGY

# Learning Outcomes

- Covers LO2 and LO3 for Module
- On completion of this lecture, students are expected to be able to:
    - work with version control systems
    - organize the features of the project separately

# CONTENT

- What is Version Control?
- Why do we need version Control?
- Introduction to GIT

# What is Version Control?

- It is the management of changes to documents, computer programs, websites or other collection of information.

- It is also known as **Revision Control** or **Source Code Management**.

# Why do we need version Control?

- How do we keep track of different versions of our code?
- How do we collaborate with other people?
- How do we make sure that if we make a change and that change doesn't work that we have an easy way to go back to older versions of our code and make sure that we have access to those as well?

**So all of that falls under the larger umbrella of what's called version control.**

There are number of different version control software that people are using on a daily basis, but one of the most popular is a piece of software called Git, which is a version control tool that we're going to be using in this lecture.

- Other Tools : Tortoise SVN

# Introduction to GIT

First thing that Git is good at doing is **keeping track** of different changes to your code. So, if you're making changes, you not only have access to the latest version of your code, but also if you start out with a particular file and then you make changes and add another line in it, or if you remove a line in it, you can **keep track of all of those different versions** of those files. This means that you know the history of your project and how it's developed from the beginning to all the way up until where you are now.

# Terms:

- **Remote Repository** – Git repository hosted on the Internet or some other network.
- **Clone** – used to make a local copy of the remote repository.
- **Local Repository** – the output of cloning a remote repo.
- **Branch** – represents an independent line of development and contains a unique set of code changes. Each repository can have one or more branches.

# Terms:

- **Commit** - captures a snapshot of the project's currently staged changes.
- **Pull** - fetches and merges changes on the remote repo to your local repo.
- **Push** - an act of transferring commits from your local repository to a remote repository. Pushing is capable of overwriting changes; caution should be taken when pushing.

# Note :

Every developer should aim to have his local repository almost always in sync with the remote repo to have up-to-date code and to reduce chances of conflicts and issues raising up later on.

# TYPICAL WORKFLOW

After doing your changes to the files and editing them ....

- Commit (with an appropriate message)
- Pull (to get others' changes newly added to the branch if any)
- Push (to send your work and update the remote branch)

# BRANCHES

- ❑ Don’t break your master.
- ❑ At any given time, all members should be working on something other than master branch. Only push code into the master when you have something stable.

Branch types:

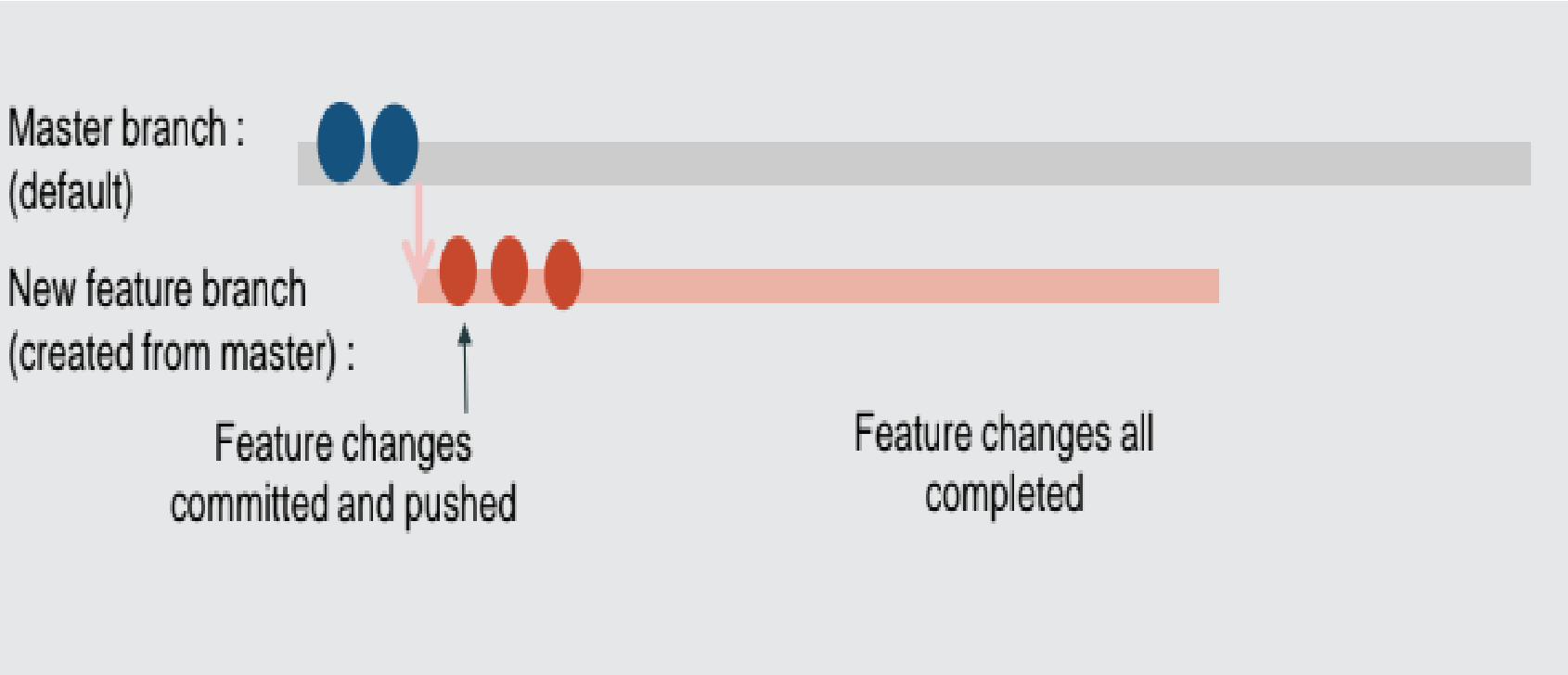
- ❑ Feature branches
- ❑ Develop / Integration / Staging branches
- ❑ Personal branches?

# 1. FEATURE BRANCHES

Create a list of implementation features for your project (on trello?)

- ❑ One dedicated branch for a feature
- ❑ Some of the members can work on one feature branch while others can work parallely on other feature branches.
- ❑ Use a branch naming convention (feature name/id?)
- ❑ feature/feature-01

# Creating Feature Branches :



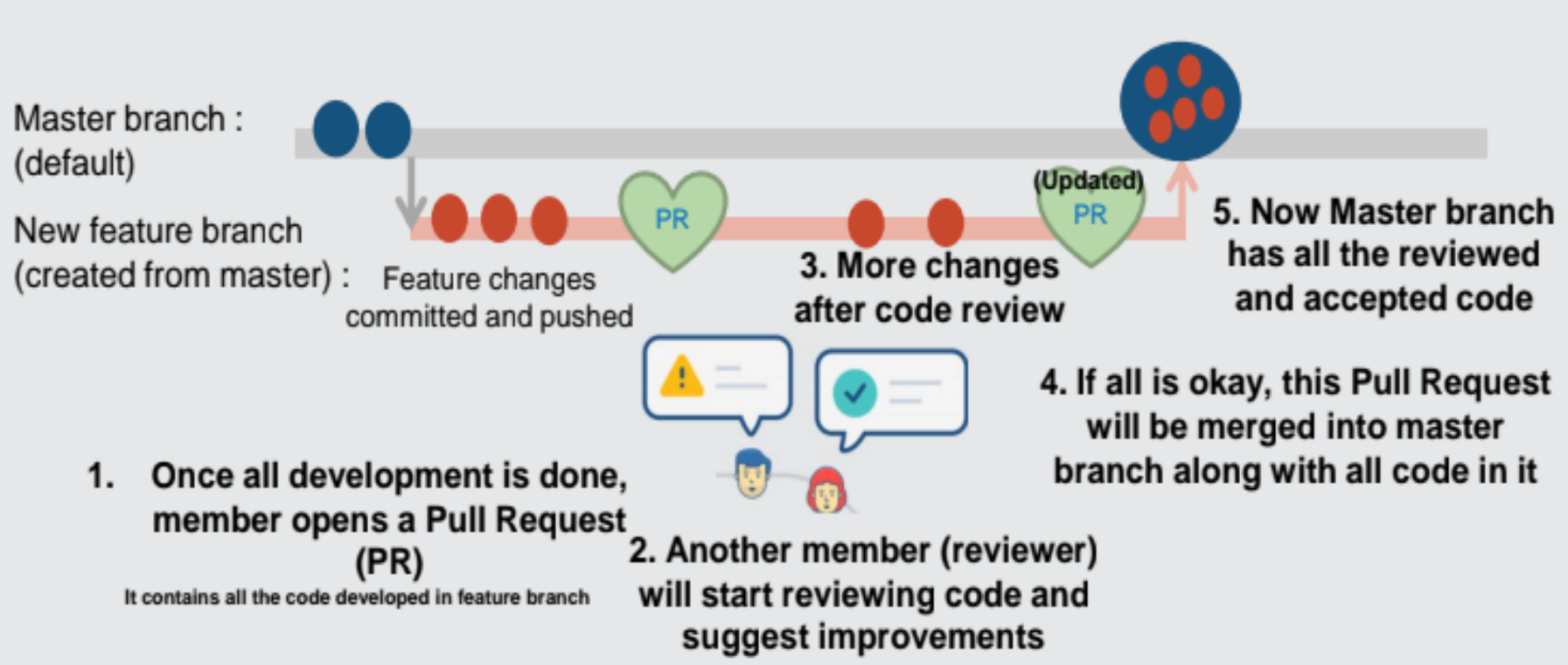
# What Next ?

SO ONCE DEVELOPMENT OF THE FEATURE IS DONE WHAT SHOULD BE DONE?

- Need to merge the changes from the feature into the Master branch.

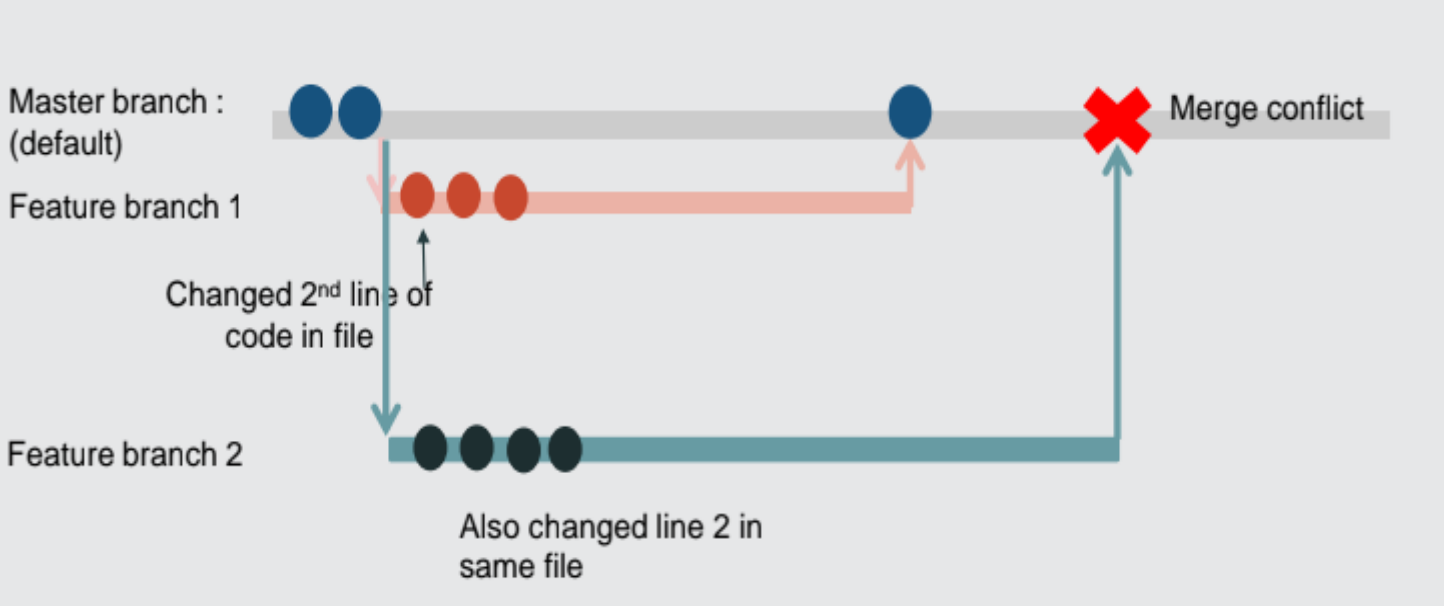
# Pull Requests ?

# WORKFLOW WITH PULL REQUESTS



# Merge Conflicts

A conflict arises when two separate branches have made edits to the same line in a file, or when a file has been deleted in one branch but edited in the other.



# Solution :

- You need to manually resolve the merge.

# REDUCE MERGE CONFLICTS: A FEW TIPS

- **Start your day by staying current** :
  - Always get the latest code, the most current (from where you branched off from). Pull changes into working branch as often as you can.
- **Create a good architecture in the project where members can work on different files rather than the same file:**
  - (decreases the probability of editing the same line of code)
- **Do not beautify a code outside of your changes:**
  - (decreases the probability of editing the same line of code)
- **Don’t leave Pull requests out for long:**
  - Reduces chances of conflicts the sooner it is merged into master.

# BEST PRACTICES:

Make sure code is committed using your correct name.

- ❑ Use a .gitignore file (if required).
- ❑ Use a branch naming convention (feature name/id?)

  **feature/feature-01**
- ❑ Branches are better short-lived. So are Pull Requests in an open state.
- ❑ If you want to move your current uncommited changes somewhere safe use the ‘Stash’ feature.
- ❑ It will save them for you until you want it back (unstash it and all the files will come back to your working directory)
- ❑ Add good commit messages
- ❑ Don’t be vague (eg: “Fixed bugs”)
- ❑ Include identifiers in commit message to keep track of commits

# REFERENCES


- https://www.specbee.com/blogs/git-best-practices-how-make-most-git
- https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell
- https://nvie.com/posts/a-successful-git-branching-model/
- https://martinfowler.com/articles/branching-patterns.html
- https://www.openbankproject.com/how-to-avoid-merge-conflicts-on-git/
- https://www.toptal.com/developers/gitignore
- https://www.datree.io/resources/github-best-practices