netmind
a bts company

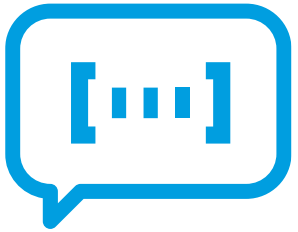
06

CI/CD - Source Code Management

netmind
a bts company

Index

netmind
a bts company



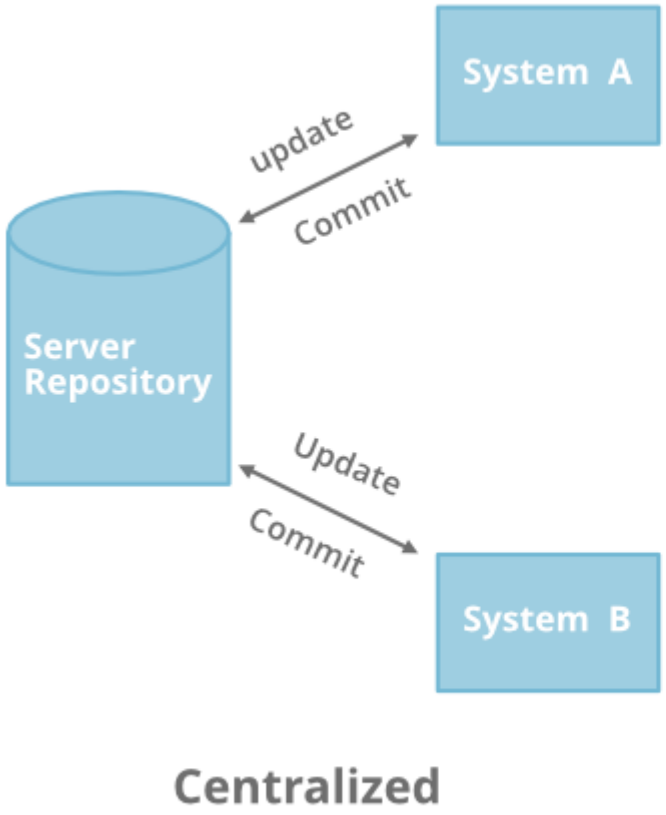
Source code management

01

Moving toward continuity

- SCM is the first stage in the CI/CD pipeline.
- Source code management (SCM) is **used to track modifications to a source code** repository.
- **All developers contribute** commiting their code into common repository, where it is merged
- There are **two main types** of SCM:
  - Centralized
  - Distributed

Centralized vs Distributed SCM



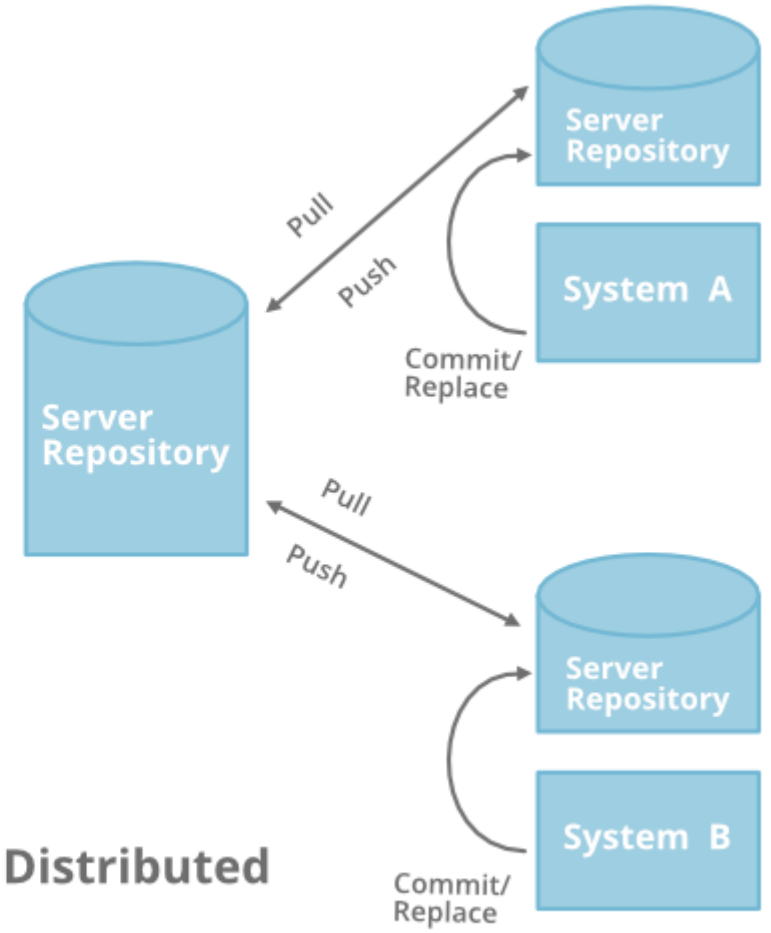


Image Source: geeksforgeeks.or

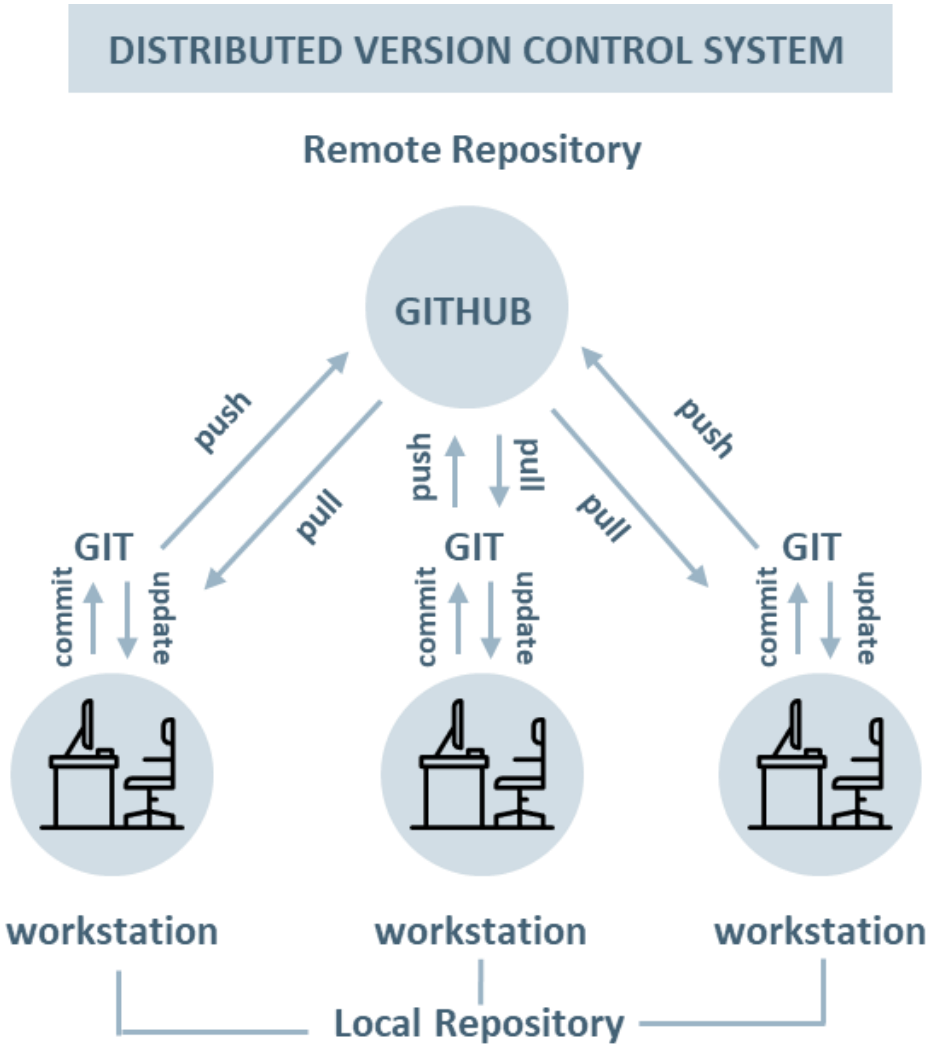
Examples: CVS or Subversion

Examples: GIT, Bazaar or Mercurial

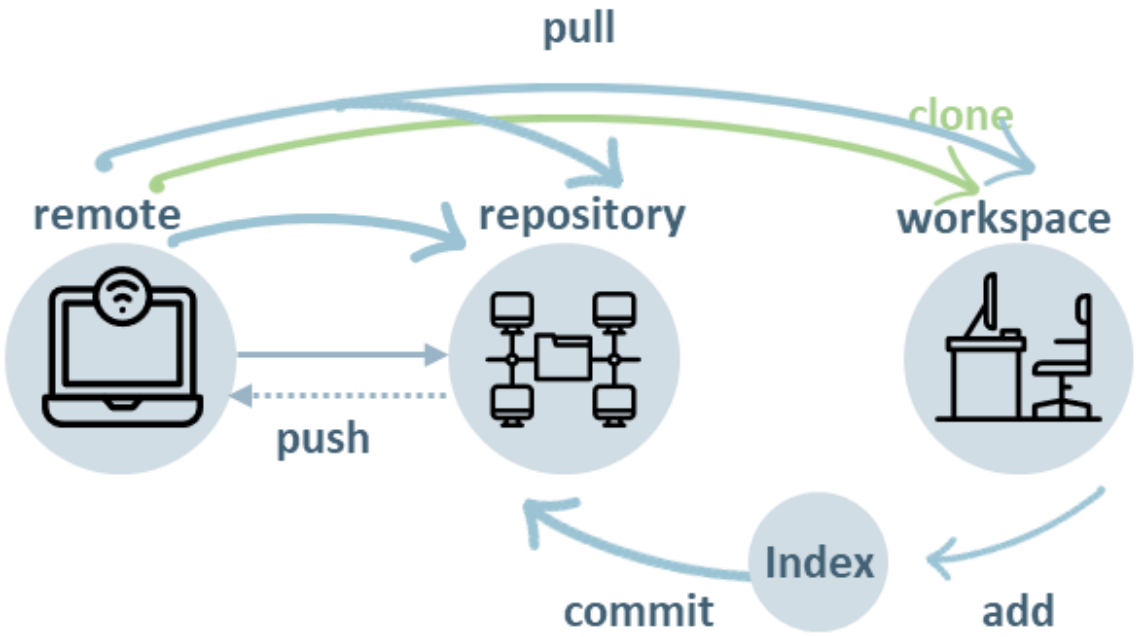
Git

- Git is version control software designed by **Linus Torvalds**, which focuses on the **efficiency and reliability** of maintaining versions of applications when they have a large number of source code files.
- Git **has a distributed architecture**, i.e. a DVCS (Distributed Version Control System): in Git the **local working copy** of all developers is also a repository that can contain the complete history of all changes.

Git repositories explained

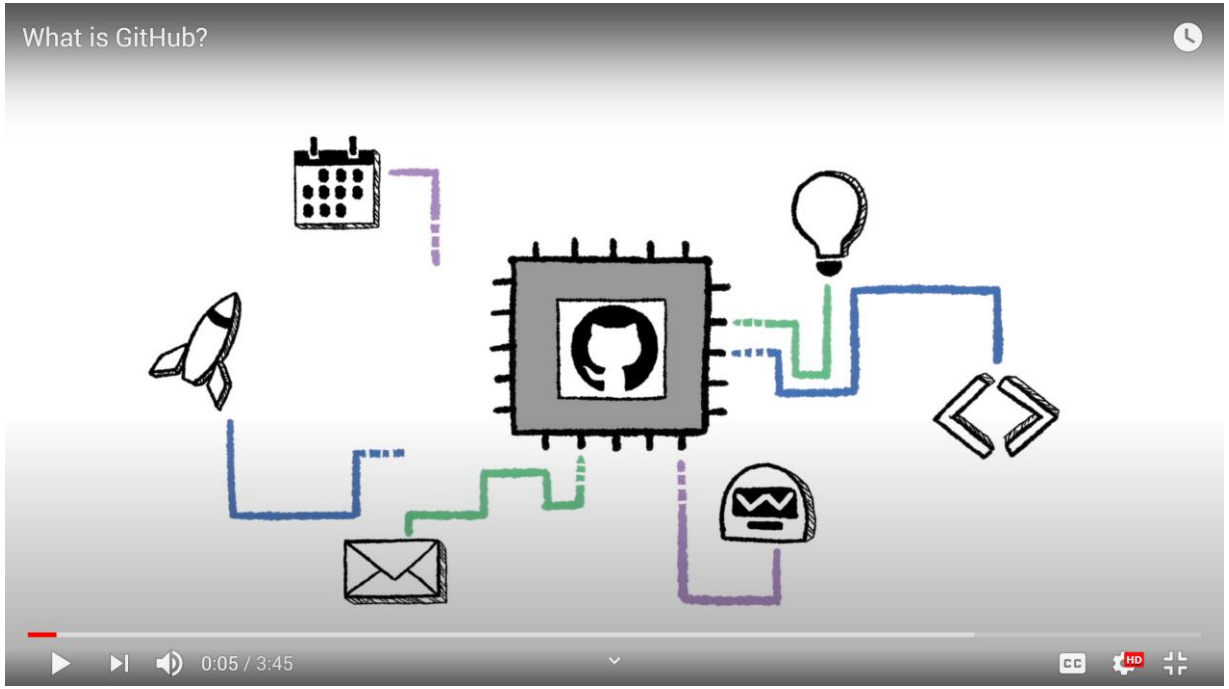


What does a typical Git workflow look like?



1. **Create** a remote repository
2. **Clone** it to your local workspace
3. **Make** changes and **Add** to the index
4. **Commit** changes to local repository
5. **Push** changes to remote repository

GitHub



https://www.youtube.com/watch?v=w3jLJU7DT5E

1. GitHub is a **project management and code version control system**, as well as a social networking platform designed for developers (https://github.com/)
2. Its **strengths** include:
   - Bug tracking.
   - Quick search.
   - It has a strong community of developers around the world.
   - Allows the source code to be downloaded as a file.
   - Enables import into Git, SVN or TFS.
   - You can customise any host service in the cloud.

OBJECTIVE

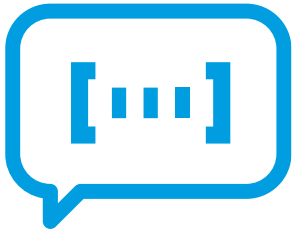
GitHub account

INSTRUCTIONS

1. Access github.com and create an account for yourself.

5 min

netmind
a bts company



Using Git

02

Git commands at a glance

Extract from: Atlassian.com

Here is a list of some basic commands to get you going with Git...let's take some minutes to review them:

Git task	Notes	Git commands
Tell Git who you are	Configure the author name and email address to be used with your commits.Note that Git strips some characters (for example trailing periods) from user.name.	`git config --global user.name "Sam Smith"git config --global user.email sam@example.com`
Create a new local repository		`git init`
Check out a repository	Create a working copy of a local repository:	`git clone /path/to/repository`
	For a remote server, use:	`git clone username@host:/path/to/repository`
Add files	Add one or more files to staging (index):	`git add <filename> git add *`
Commit	Commit changes to head (but not yet to the remote repository):	`git commit -m "Commit message"`
	Commit any files you've added with git add, and also commit any files you've changed since then:	`git commit -a`
Push	Send changes to the master branch of your remote repository:	`git push origin master`
Status	List the files you've changed and those you still need to add or commit:	`git status`
Connect to a remote repository	If you haven't connected your local repository to a remote server, add the server to be able to push to it:	`git remote add origin <server>`
	List all currently configured remote repositories:	`git remote -v`

Git commands

Extract from: Atlassian.com

Git task	Notes	Git commands
Branches	Create a new branch and switch to it:	`git checkout -b <branchname>`
	Switch from one branch to another:	`git checkout <branchname>`
	List all the branches in your repo, and also tell you what branch you're currently in:	`git branch`
	Delete the feature branch:	`git branch -d <branchname>`
	Push the branch to your remote repository, so others can use it:	`git push origin <branchname>`
	Push all branches to your remote repository:	`git push --all origin`
	Delete a branch on your remote repository:	`git push origin :<branchname>`
Update from the remote repository	Fetch and merge changes on the remote server to your working directory:	`git pull`
	To merge a different branch into your active branch:	`git merge <branchname>`
	View all the merge conflicts:View the conflicts against the base file: Preview changes, before merging:	`git diffgit diff --base <filename>` `git diff <sourcebranch> <targetbranch>`
	After you have manually resolved any conflicts, you mark the changed file:	`git add <filename>`
Tags	You can use tagging to mark a significant changeset, such as a release:	`git tag 1.0.0 <commitID>`
	CommitId is the leading characters of the changeset ID, up to 10, but must be unique. Get the ID using:	`git log`
	Push all tags to remote repository:	`git push --tags origin`
Undo local changes	If you mess up, you can replace the changes in your working tree with the last content in head:Changes already added to the index, as well as new files, will be kept.	`git checkout -- <filename>`
	Instead, to drop all your local changes and commits, fetch the latest history from the server and point your local master branch at it, do this:	`git fetch origin git reset --hard origin/master`
Search	Search the working directory for foo():	`git grep "foo()"`

OBJECTIVE

Practice with the commands

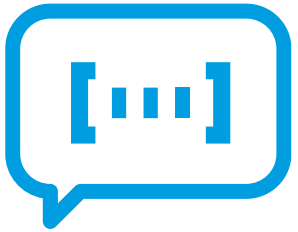
INSTRUCTIONS

1. Follow the exercises in the next repository:
   - https://github.com/ricardoahumada/git-exercises/

2. Follow the proposed exercises.

120 min

netmind
a bts company



Source Code Management practices

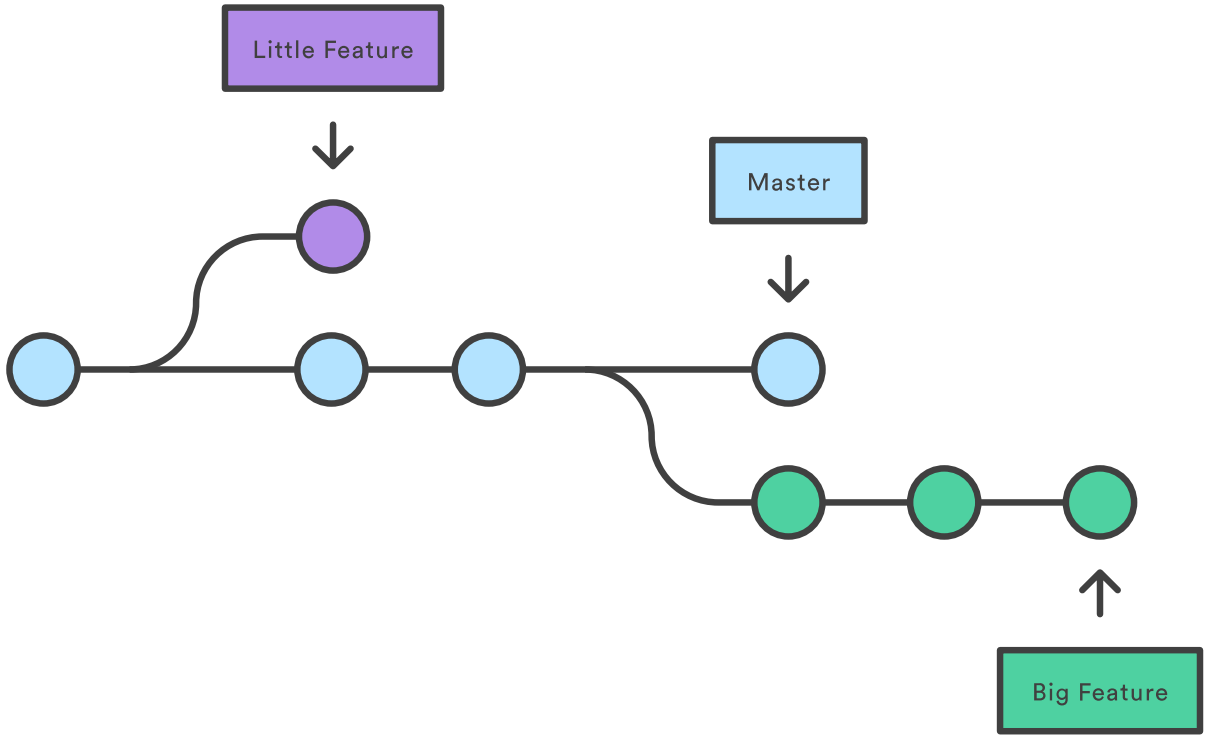
03

Best practices

- **Commit** often
- Ensure you're working from **latest version**
- Make **detailed notes**
- **Review changes before committing**
- Use **Branches**
- **Agree** on a Workflow

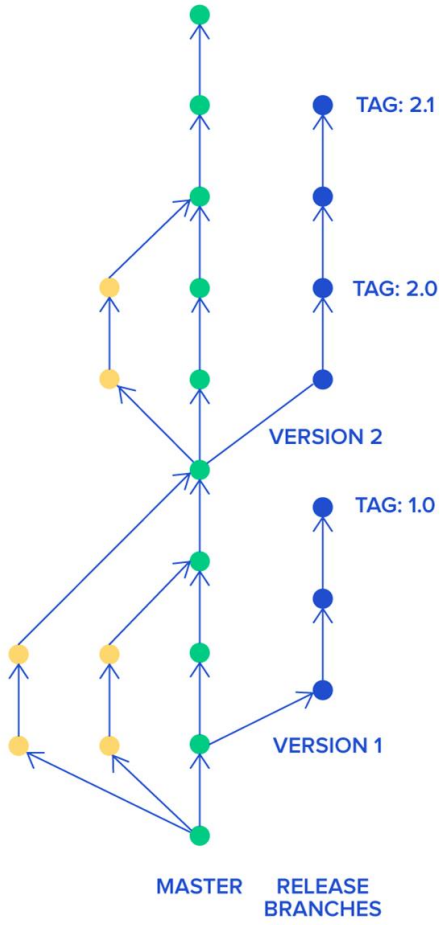
Branching

- A branch represents **an independent line of development**.
- Branches serve as an abstraction of the **edit / stage / commit** process
- In git branches are part of the daily development process: they are a **pointer to a snapshot of changes**.
- When you want to **add a new function or correct an error**, no matter how big or small it is, **a new branch is generated** to encapsulate these changes.

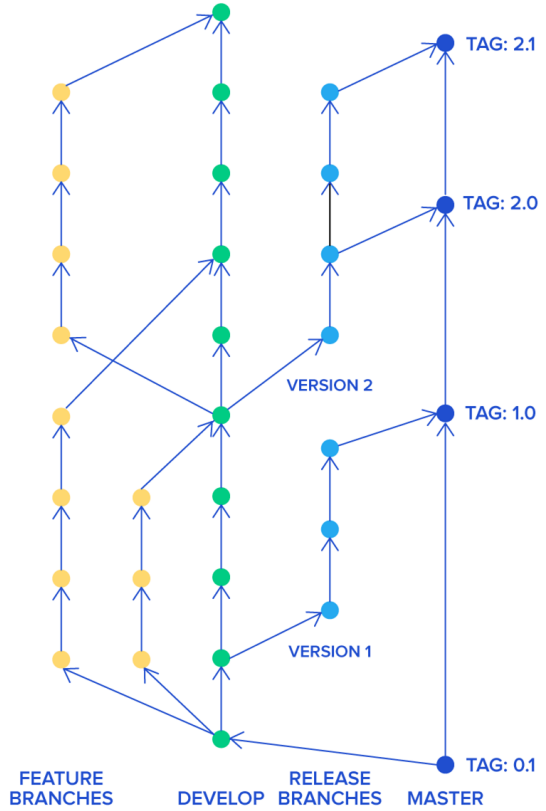


Branching workflows

Long-Running Branches: Trunk-based Development Workflow

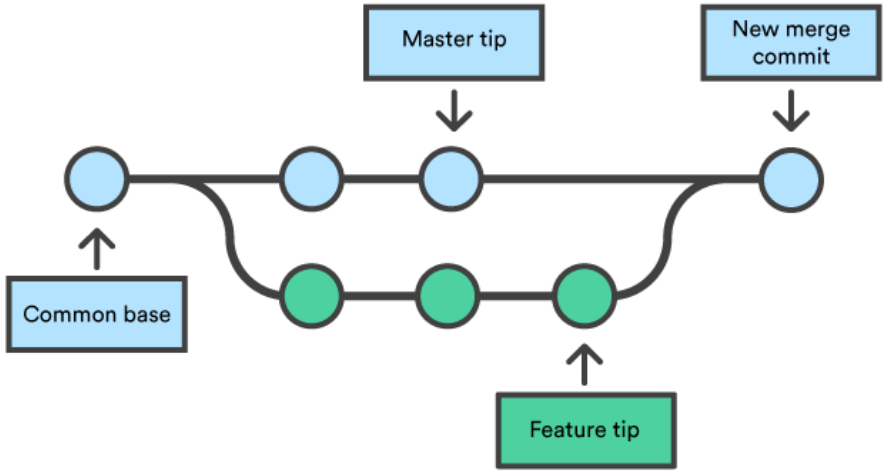


Topic Branches: Git Flow



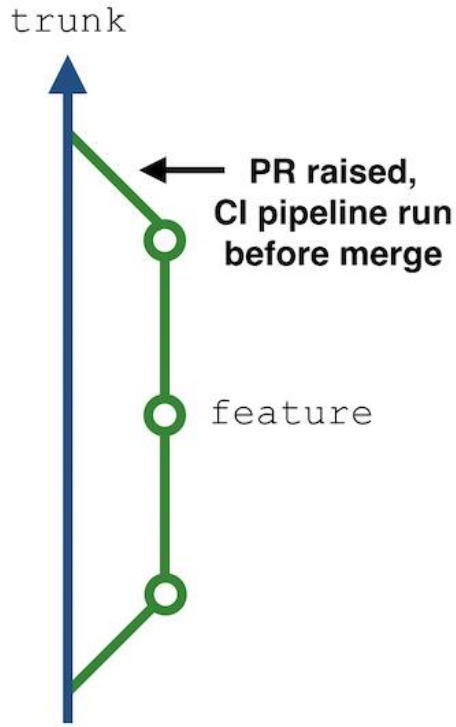
Branch merging

- Merge is the way git **rejoins a forked story**.
- The **git merge command** allows you to take the separate development lines created by the git branch and integrate them into a single branch.
- Merging allows some good coding practices as **“pull request”** and **“pair programming”**



CI/CD branching strategy: Streamline

- Streamline is a combined approach for branching, continuous integration, and (optionally) continuous deployment.
- Streamline **workflow** consist of:
    1. **Limit branches to feature and trunk** (equivalent to master). bugfix is optional.
    2. **Use pull requests** to trigger a CI pipeline that runs all functional tests.
    3. **A fast-forward merge** into trunk marks a new 'version' of your app.
    4. The **new versioned can be deployed anywhere**, from 'demo' or 'penetration test' environments right through to production.



Pull Request

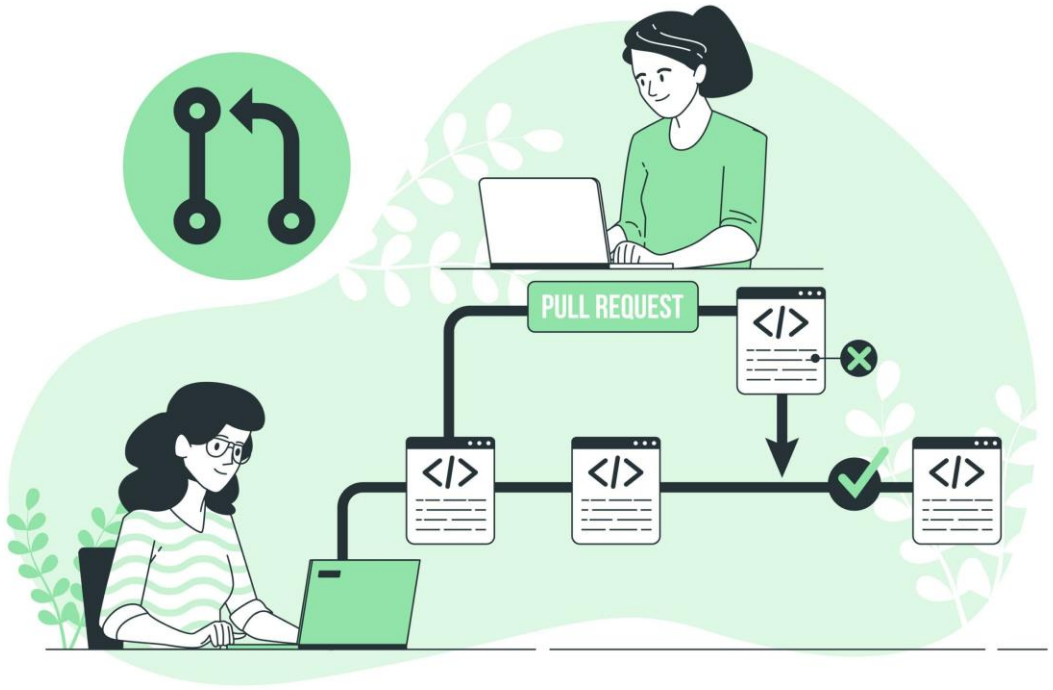
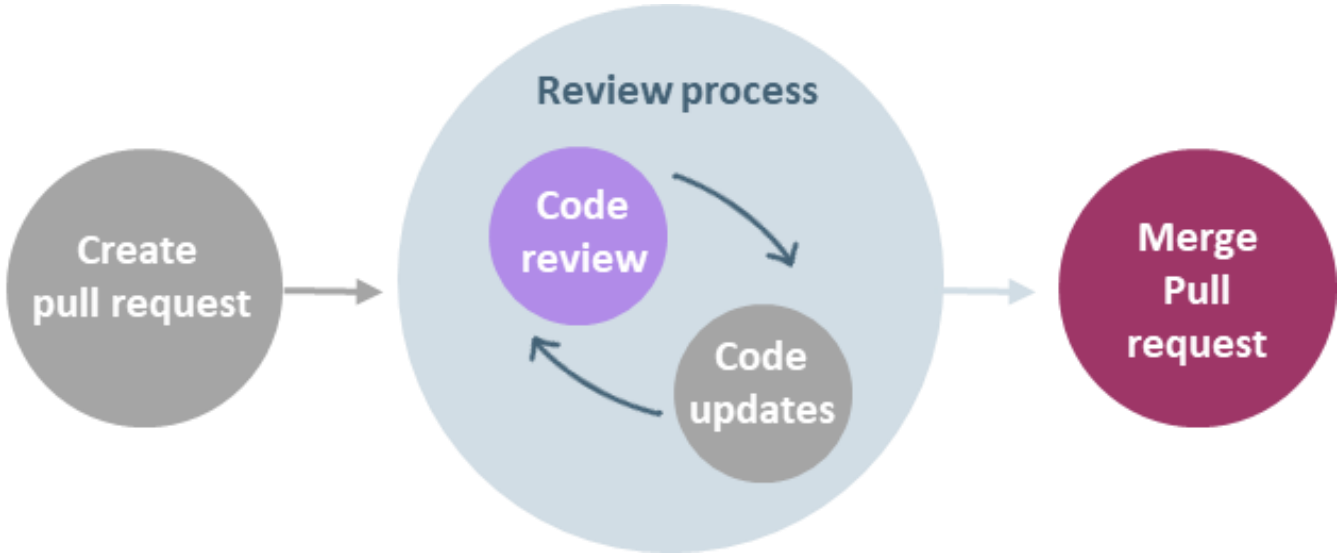


Image source: gitential.com

1. A Pull Request is a **method of submitting contributions to a project**.
2. In their simplest form, pull requests are a mechanism for a **developer to notify team members that they have completed a feature.**
3. Once their feature branch is ready, the developer **files a pull request via their repository mecanisms**. This lets everybody involved know that they need to review the code and merge it into the master branch.
4. **It is a workflow method**; not a feature of the version control system. It allows a constant code review that improve the quality of the code and contributes to teaming up.

Pull Request process



1. A developer **creates the feature** in a dedicated branch in their local repository.
2. The **developer pushes the branch** to a central repository.
3. The **developer submits a pull request**.
4. The rest of the team **(reviewers) review the code, discuss it and modify it**.
5. The **project manager merges the feature** into the official repository and closes the pull request.

OBJECTIVE

Let’s play Pull requesting

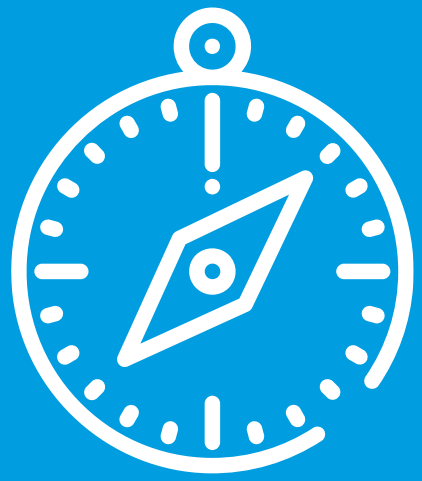
INSTRUCTIONS

1. Pair with your class partner.
2. Share a repository code (with some text files).
3. One will play the developer and the other will play the reviewer. Follow next slide process.
4. Then interchange the role and repeat the next process.

30 min

Pull request process:

- Developer must:
  1. Generate a feature branch
  2. Make a change..then add, commit and push the change.
  3. Raise a pull request in Github.
- Reviewer must:
  1. Accept the pull request.
  2. Review the changes.
  3. Make some comments and suggestion to the developer (via Github and face to face).
- Developer must:
  1. Read/listen the comments and suggestion from reviewer.
  2. Adjust the code and push it again.
- Reviewer must:
  1. Accept the changes and merge the branch.



Next steps

netmind
a bts company

Thanks!

Follow us: