Git Fundamentals

Mike Pua, Software Engineer
Orange & Bronze Software Labs

Objectives

At the end of this session, you will be able to

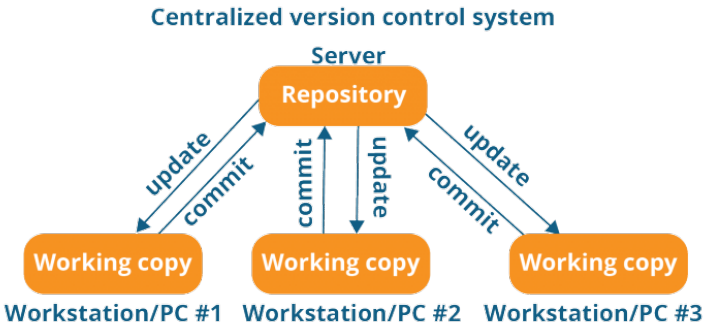
- understand basic version control concepts with Git
- familiarize with Git workflow
- branch, merge and use remote repositories

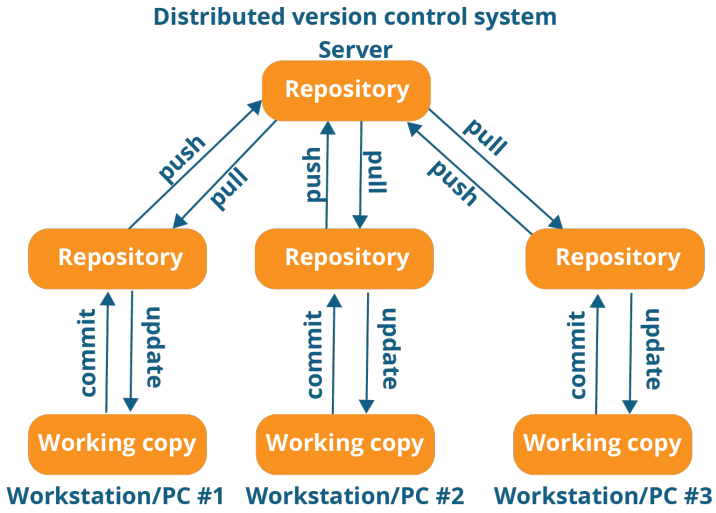
What is Git

https://git-scm.com

- Free **distributed version control system** (VCS).
  - No central repository.
  - Avoids single point of failure.
- Invented by Linus Torvalds in 2005
- Embraces non-linear development.
  - Users are encouraged to branch and merge eagerly.
- Each commit is a snapshot of an entire project, not of individual files.

Git Fundamentals
Centralized vs Distributed



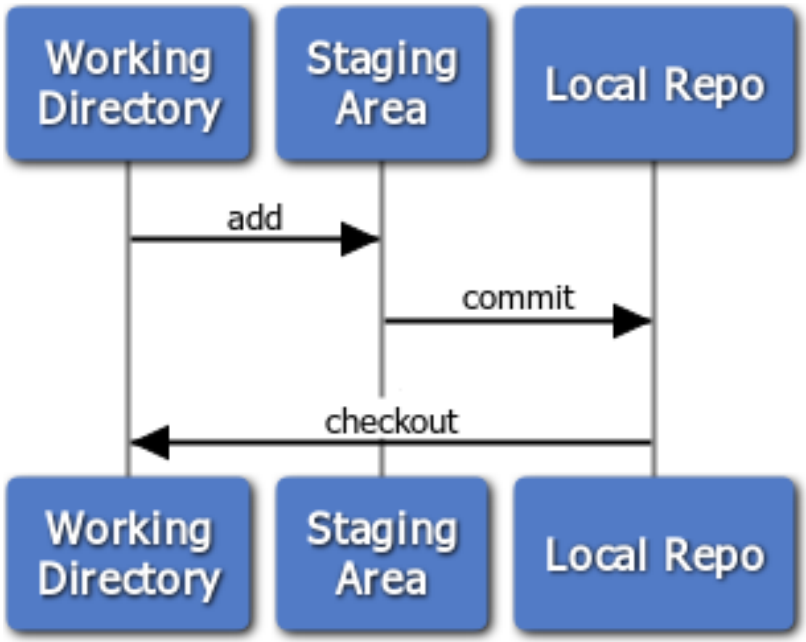


4/27

Git Fundamentals

The Three Areas, Three States

- **Modified / Untracked**
  - Untracked files are new to the Git repository
  - Modified files were previously committed to Git and has altered
- **Staged**
  - Indexed files
  - Files and revisions that are ready to be committed
- **Committed**
  - Committed files are safely stored in Git repository



5/27

Git Fundamentals

- **`git init`**
  - Turns a folder into a git repository
- **`git status`**
  - Displays the state of the working directory and the staging area
- **`git log`**
  - Shows history of the repository
  - most recent is displayed first
  - More Samples:
    - `git log --oneline`
    - `git log --author="<author name>"`
    - `git log -before <date>`
    - `git log --before 2019-01-01 --oneline`

Git Fundamentals

- **`git add`**
  - Adds file (or files) to staging area (git index)
  - `git add <file-path>`
  - `git add .`
- **`git commit`**
  - Saves your changes to Git repository
  - stores a commit object that contains a pointer to the snapshot of the content you staged
  - Uncommitted files are not part of Git's database
  - Is the commit message required? Yes.
  - `git commit -m "<commit message>"`

Git Fundamentals

- **`git stash`**
  - Use when you have to save your changes, but does not want to commit them yet and you need a clean working directory
  - `git stash`
  - `git stash save "<Describe stash>"`
  - `git stash list`
  - `git stash apply`
  - `git stash pop`
  - `git stash clear`
  - `git stash drop stash@{1}`
  - `...`
- **.gitignore**
  - update this config file to ignore certain files and directories
  - normally tracked by Git as regular file

Git Fundamentals

- **`git config`**
  - used to view and set Git configurations
  - Displaying setting values
    - `git config user.name`
    - `git config user.email`
    - `git config --global user.name`
    - `git config --global user.email`
  - Setting values
    - `git config user.name "Mike Pua"`
    - `git config user.email "mike@ust.edu.ph"`
    - `git config --global user.name "Mike Pua"`
    - `git config --global user.email "mike@ust.edu.ph"`
    - `...`

Git Fundamentals

10/27

Git Fundamentals

- **`git gui`**
  - Shows the graphical tool for crafting commits
- **`gitk`**
  - Displays the Git repository browser
  - `gitk --all`
  - `gitk --author="Mike Pua"`
  - `gitk --author="mike.pua@orangeandbronze.com"`

Git Fundamentals

- **`git tag`**
  - Used to create, list, delete tags
  - Used to tag specific points in a repository's history
  - `git tag v1.0.0`
  - `git tag -a v1.0.1`
  - `git tag -a v1.0.2 -m "Releasing version v1.0.2"`
  - `git tag -d v1.0.0`
  - `git show v1.0.1`

Git Fundamentals

- **`git branch`**
  - Used to list, create, or delete branches
  - Git uses master as `default` branch
  - A branch is a pointer to a commit (not a container for commits)
    - https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell
  - `git branch new_branch`
  - `git branch -d feature-list-students`
- **`git checkout`**
  - Used to switch between branches, between commits or between files
  - `git checkout feature-list-students`
  - `git checkout <commit ID>`
  - `git checkout <file location>`
  - `git checkout -b feature-login-screen`

Git Fundamentals

- **`git revert`**
  - reverts the effect of earlier commit/s
  - `git revert <commit ID>`
  - `git revert --no-commit HEAD~3..`
    - reverts last 3 commits as one commit (This will not work if some commits are merge commits)
    - The dots specify a range. HEAD~3.. is the same as
    - Same as `git revert --no-commit HEAD~3..HEAD`
    - HEAD is a reference to the current (checked out) commit your are in
    - HEAD is a pointer that points to the current branch
- **`git merge`**
  - Joins histories together
  - `git merge <branch_name>`
    - Merges `branch_name` to the current branch

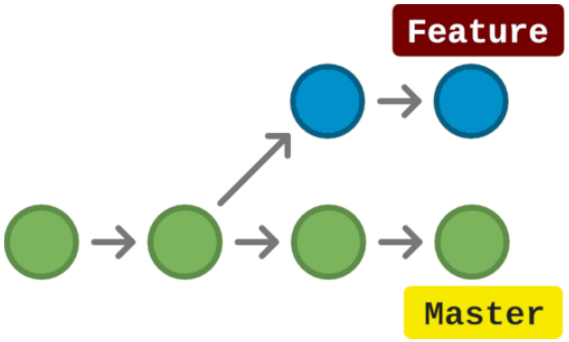
Git Fundamentals

- **`git mergetool`**
  - Runs tool for merge conflict resolution
  - During merge, if Git is able to resolve differences, it automatically joins the revisions being merged.
  - When Git cannot resolve merge conflicts you must
    - edit the file (e.g., using a mergetool),
    - stage the changes, then
    - commit the staged changes
- **`git rebase`**
  - changes local history by moving or combining a sequence of commits to a new base commit
  - `git rebase master new_feature`
  - must be done for local commits only as rebase modifies the history

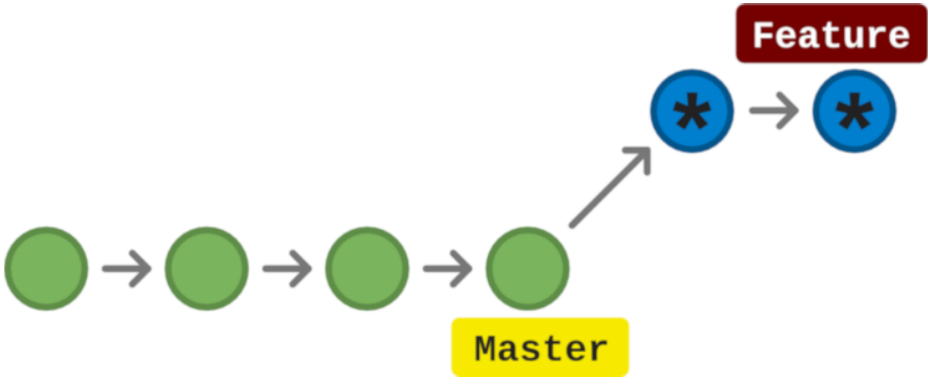
Git Fundamentals

Merge vs Rebase

Before



After `git rebase Master Feature`



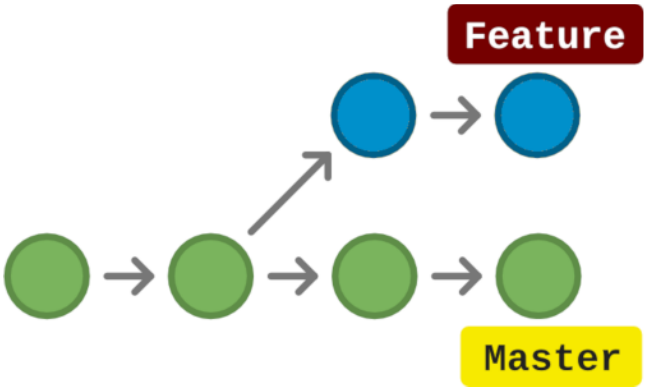
- Rebase recreates the new commits
- History is modified
- History looks clean & flat

16/27

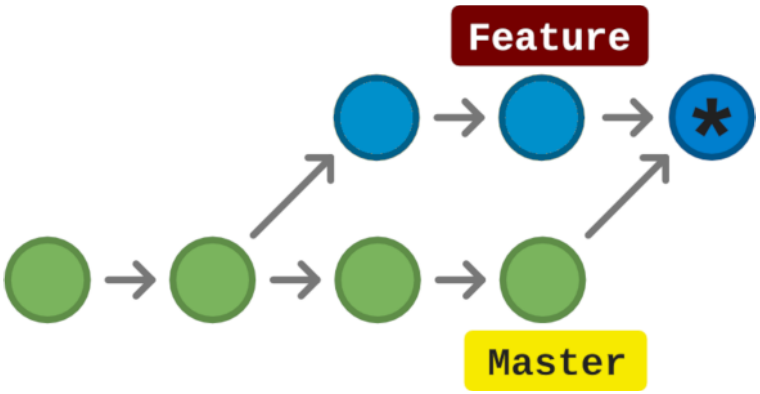
Git Fundamentals

Merge vs Rebase

Before



After `git merge Master Feature`



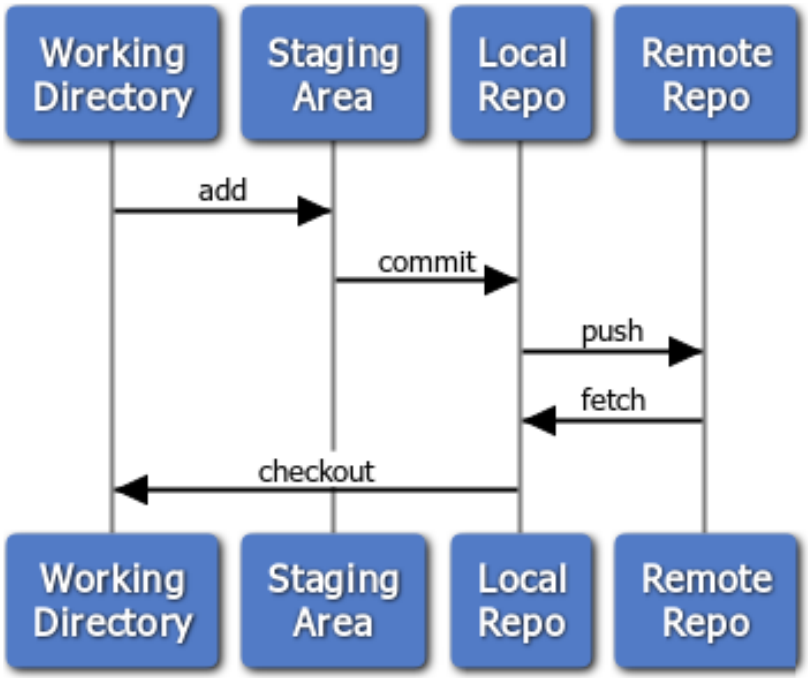
- Merge is recorded as a new commit
- History is intact (not modified)
- History can be polluted by merge commits

17/27

Git Fundamentals

Git Fundamentals
Repositories

- **Local repository**
  - the repository or Git database where you commit your changes to
  - the `.git` folder in your working directory
  - has a working directory
- **Remote repository**
  - the repository located on a remote server on the internet or in your local network
  - used by team members to share & exchange data
  - has no working directory



Git Fundamentals
Working with Remote Repository

- **`git clone`**
  - creates local copy of an existing remote repository
  - creates a new folder on the current folder you are in and downloads the Git database
  - `git clone <project_url>`
- **`git fetch`**
  - downloads new data from remote repository
  - safe way to download and review commits as it does not affect the current working directory
- **`git pull`**
  - updates the current branch with the latest changes from the remote server
  - combination of `git fetch` and `git merge`
  - can encounter a merge conflict
    - Better run `git pull` only with a clean working copy

Git Fundamentals
Working with Remote Repository

- **`git push`**
  - applies your local commits to the remote repository
  - `git push origin master`
    - will push to remote `master` branch
  - `git push -u origin new_branch`
    - will push the new branch and its relevant commits
    - `origin` is the alias of the URL of the remote repository
    - `-u` makes new_branch branch track the remote branch of the same name
  - `git push --tags`
    - pushes tags; tags are not pushed unless this is used
  - `git push -f`
    - Imposes local commit history over that of the remote repository (**Use with caution**)

Git Fundamentals

- **`git blame`**
  - Displays the last person who changed each line
  - `git blame <filename>`
  - git blame filename -L 0,10
    - shows commiter responsible for changes from line "0" to line "10"

Git Fundamentals

Best Practices

- Commit early, commit often.
- Make commits small & atomic
- Descriptive commits properly.
- Pull & merge your teammates' work early & often.
  - The longer you wait, the harder to merge.
  - Run tests after every merge.
- Never push a broken build
  - Run tests before pushing.
- Push early & often.
  - Otherwise your teammates will have difficulty merging your work.
- Use a Continuous Integration tool.
  - The sooner your team detects a bug, the easier to fix.

Conclusion

In this session, we discussed about basic Git concepts

- version control,
- Git workflow,
- branching, merging, and
- use of remote repositories.

We covered only a tiny fraction of Git's capabilities.

- Continue to study on the capabilities of Git most suited for your own situation.
- Git is well-documented, with numerous tutorials and discussion forums.
- Consider the numerous tools you can use with Git to enhance your experience & productivity.

Git Fundamentals

For feedbacks, scan the QR code below

Thank You!

www.orangeandbronze.com