OVERVIEW

- Working with Remotes
- Some Essential Commands
- Common Git Workflows
- Your First Git Conflict
- Hands-on Time

Working with Remotes

A remote in Git is a common repository that all team members use to exchange their changes. In most cases, such a remote repository is stored on a code hosting service like GitHub or on an internal server.

Let’s try GitHub as an example.

Some Essential Commands

Ignoring files/folders (un-tracking paths)

.gitignore is a file allows to ignore files and directories from being added to the index. By matching patterns you tell Git not to track certain paths.

To start ignoring files already existing in the Git index, you have to first remove them from the index using `git rm --cached <path>` or `git rm -r --cached <path>`, then add them to .gitignore.

Some Essential Commands

Undoing changes before commit

If you happen to have made edits to certain files and at some point decided that you no longer need those uncommitted changes, you can use `git checkout —` or `git stash`.

`git checkout — <path>` will simply discard the changes so it's never undoable.

`git stash` would stash all of your uncommitted changes but they will be saved safely, you can then undo a stash call by calling `git stash apply`, or `git stash list` to list all saved stashes, `git stash apply <id>` to apply certain stash..

Some Essential Commands

git reset

`reset` is a useful subcommand of Git and has a variety of uses. Let's talk about a few:

- undoing `git add`
    $ git add . # adding all changes
    $ git reset # undoing add for all changed files
    Or
    $ git add file1.txt file2.txt
    $ git reset file2.txt # undo add for 1 file

- undoing `git commit`
    $ git reset HEAD^ # reset to previous commit
    $ git reset <commit-id> # reset to specific commit

- reset the index to a previous commit (hard reset)
  `git reset —hard <commit-id>`

Some Essential Commands

git rebase

When you create a branch of off another, git will set the tip of your created branch to the last commit (HEAD) of your source branch.

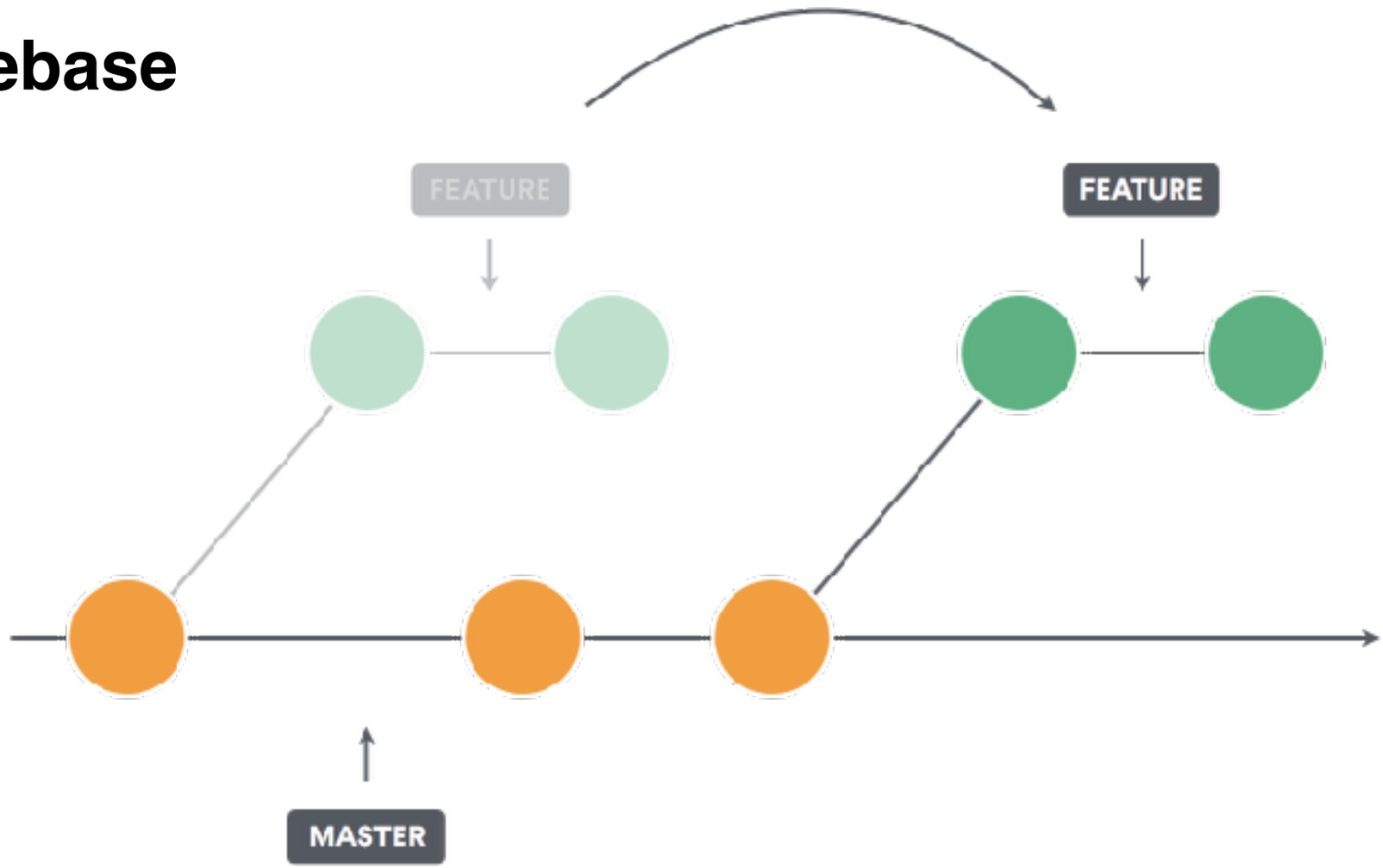
For instance when you `git checkout feature` from `master`, and `master` has commits `c1, c2, c3`, then the feature branch will be based on c3 commit, having any preceding commits.

When you continue developing the master branch, you add further commits c4 and c5, and at some point you want to to checkout back to feature branch to sync it and have it pull the recent changes (c4 and c5). This can be done via rebasing.

Being checked out to the feature branch, simple run `git rebase master` to rebase the base commit from c3 to c5. And now you have synced branches!

Some Essential Commands

git rebase



Some Essential Commands

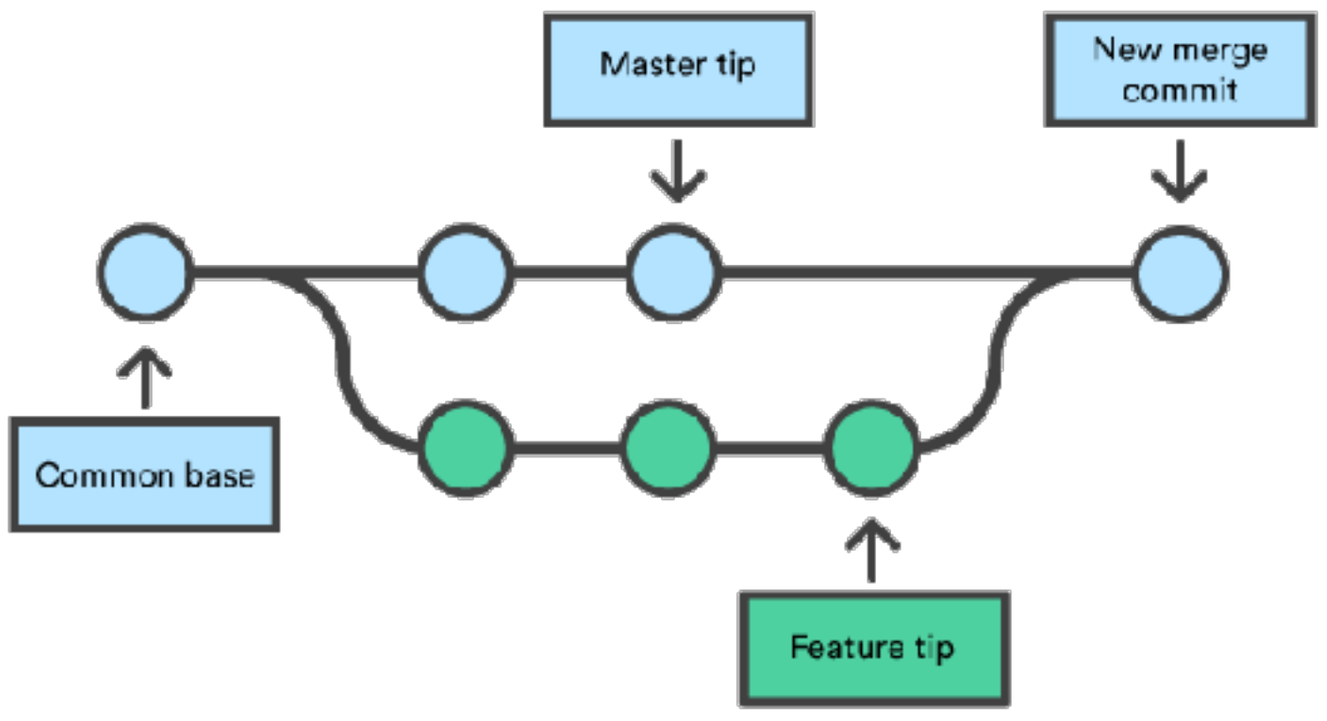
git rebase

Try out an example.

Some Essential Commands

git merge

When you are done developing a feature, you want to merge your feature branch back into master or your default branch. This process can be done via `git merge`.



Some Essential Commands

git merge: Your first Git conflict

Conflicts will often occur when you work with Git. If they don't, then you're probably doing it wrong.

A sample conflict is when you make changes to the same file but on different branches, or in the remote repo vs the local repo, and then you attempt to merge branches or pull/push changes to remotes.

There are several tools for fixing Git conflicts, we will stick to manually resolving conflicts using Git hints and a text editor.

Let's try an example.

Common Git Workflows

A Git Workflow is a recipe or recommendation for how to use Git to accomplish work in a consistent and productive manner.

When working with a team on a Git managed project, it's important to make sure the team is all in agreement on how the flow of changes will be applied.

Common Git Workflows

Basic

The most popular workflow among Git developers and the entry stage of every project.

The idea is simple: there is one central repository. Each developer clones the repo, works locally on the code, makes a commit with changes, and push it to the central repository for other developers to pull and use in their work.

Common Git Workflows

Basic

Common Git Workflows

Feature Branch

The core idea behind the Feature Branch Workflow is that all feature development should take place in a dedicated branch instead of the master branch.

Branches are independent “tracks” of developing a project. For each new functionality, a new branch should be created, where the new feature is developed and tested. Once the feature is ready, the branch can be merged to the master branch so it can be deployed.

Common Git Workflows

Feature Branch

Common Git Workflows

Feature Branch + Merge Requests

This works the same way as the “Feature Branch” workflow, except for enforcing reviews before merging a feature into master.

This is done via Merge Requests, commonly known as Pull Requests on GitHub. When a developer has a feature that is ready to be merged and deployed, they can send a merge request, it gets reviewed by the project reviewers, implementing further requested changes if necessary before merging.

Common Git Workflows

Gitflow

Basically Gitflow consists of automating your merge and deploy work so you wouldn’t end up losing hours reviewing everything every time.

This is done by integrating unit tests (TDD), integration tests, continuous integration (CI), etc into your workflow.

We will not try this as it is a bit advanced to grasp.

Hands-on Time

Try out one or more of the workflows addressed in this talk, you can use your GitHub or other Git cloud service account.

Let's also try the "Feature Branch + Merge Requests" workflow, you can submit a PR to http://github.com/elhardoum/git-workshop-19, to add your name and GitHub hello-world repository to git-workshop-19.

Slides will be available in the aforementioned GitHub repository.