Git Mini XTR

Pratik Garg

Session 1

40 minutes

Git Concepts

is free and open source distributed version control system

has a tiny footprint with lightning fast performance.

is the safest VCS I know

Every file and commit is checksummed and retrieved by its checksum when checked back out.
It's impossible to get anything out of Git other than the exact **bits you put in**.

Rocks! If you know a few concepts

if you don't…
it gets ugly fast

Git mislabels things in confusing ways

ex: git branches aren't branches

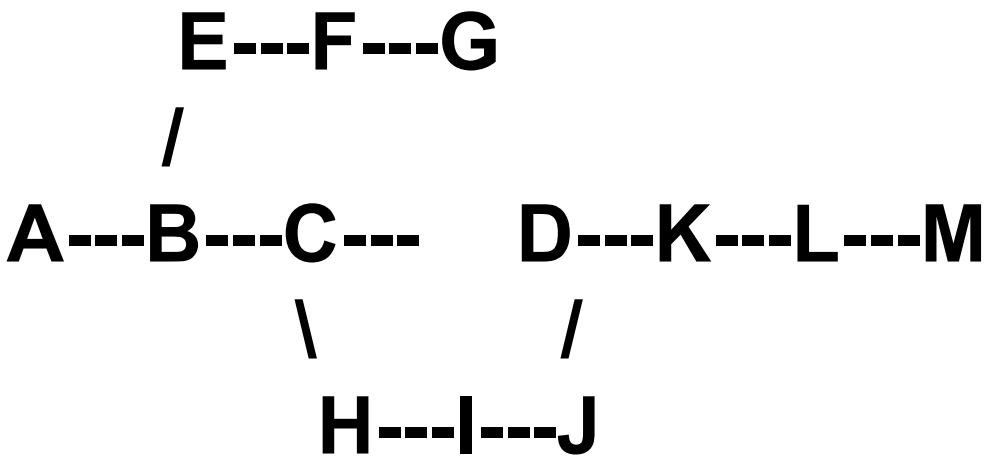
Git has hundreds of commands, but commonly used ones require extra parameters

Git uses dangerous-sounding terms:

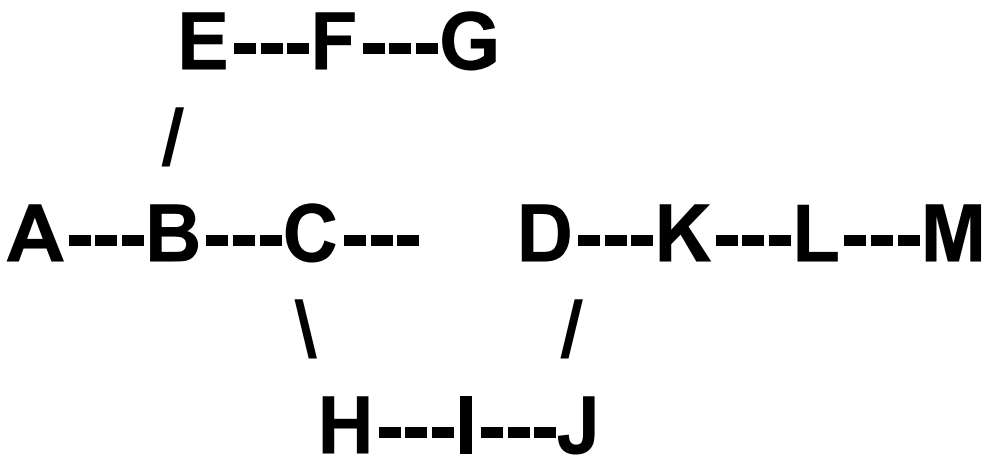
“rewrite history”
rebase
reset --hard HEAD
squash
fast-forward
reflog

Throw away your preconceptions from other version control systems

Git Repo is a DAG (Directed Acyclic Graph)



DAG nodes represent a Commit



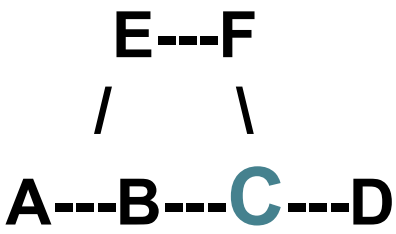
- A commit is identified by a unique SHA
- Commits are completely immutable
- You cannot modify commits, only add new ones

What points at commits?

- Child commits
- Tags
- Branches
- Reflog

Child commits

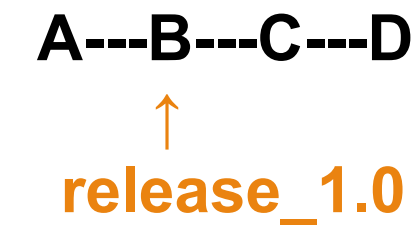
Point at 1..N parent commits



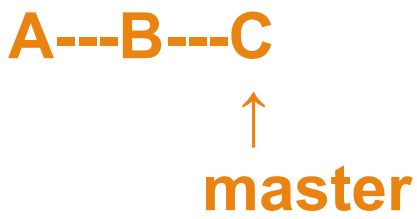
Most commonly 1 or 2 parent commits

Tags - Fixed pointers

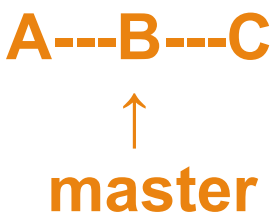
Git commit -m "adding stuff to C"



Branches - They're just pointers, and are easy to move if you don't like where they are at



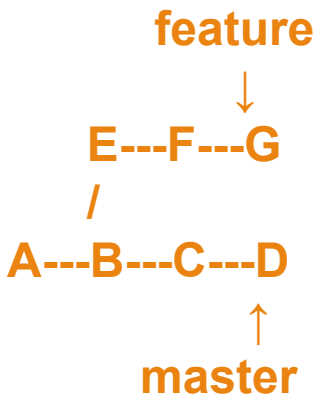
% git reset --hard SHA_OF_B



Branches

- Commits don't “belong to” branches, there's nothing in the commit metadata about branches

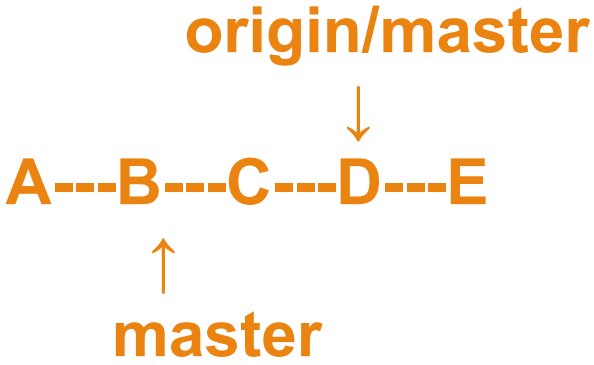
- A branch's commits are implied by the ancestry of the commit the branch points at



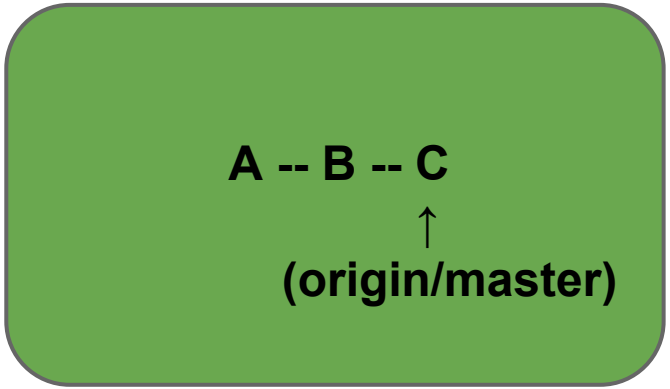
master is A-B-C-D and feature is A-B-E-F-G

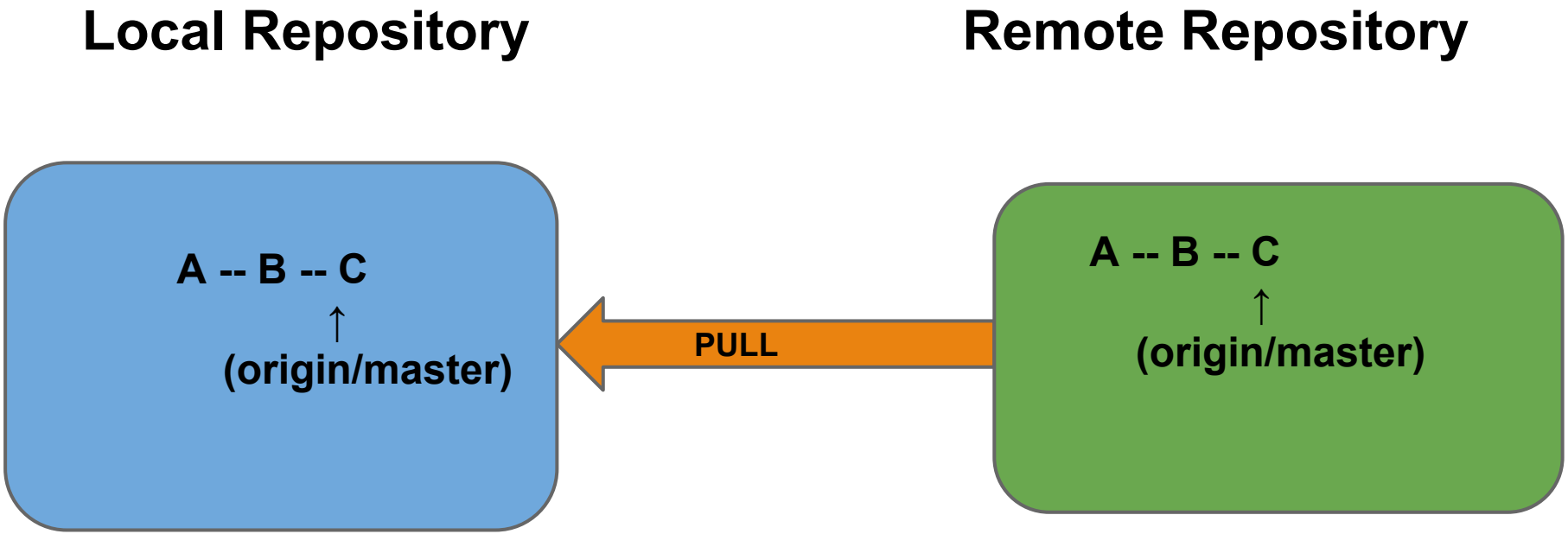
Remote branches

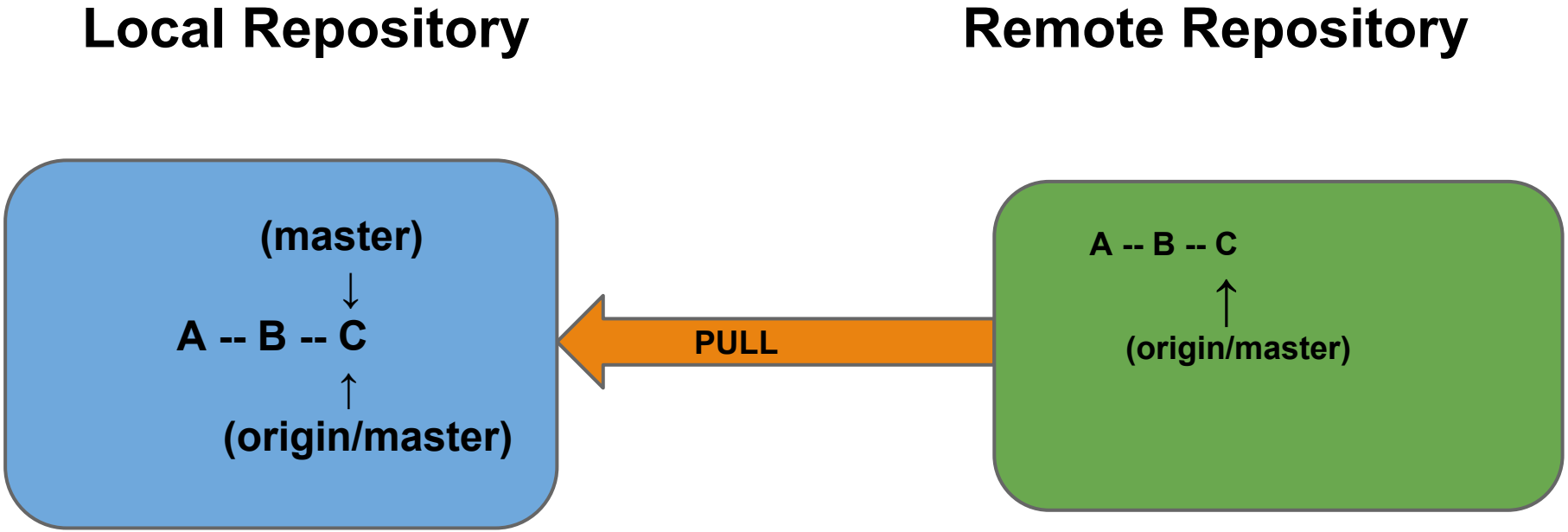
“Remote” branches are just pointers in your local repository.

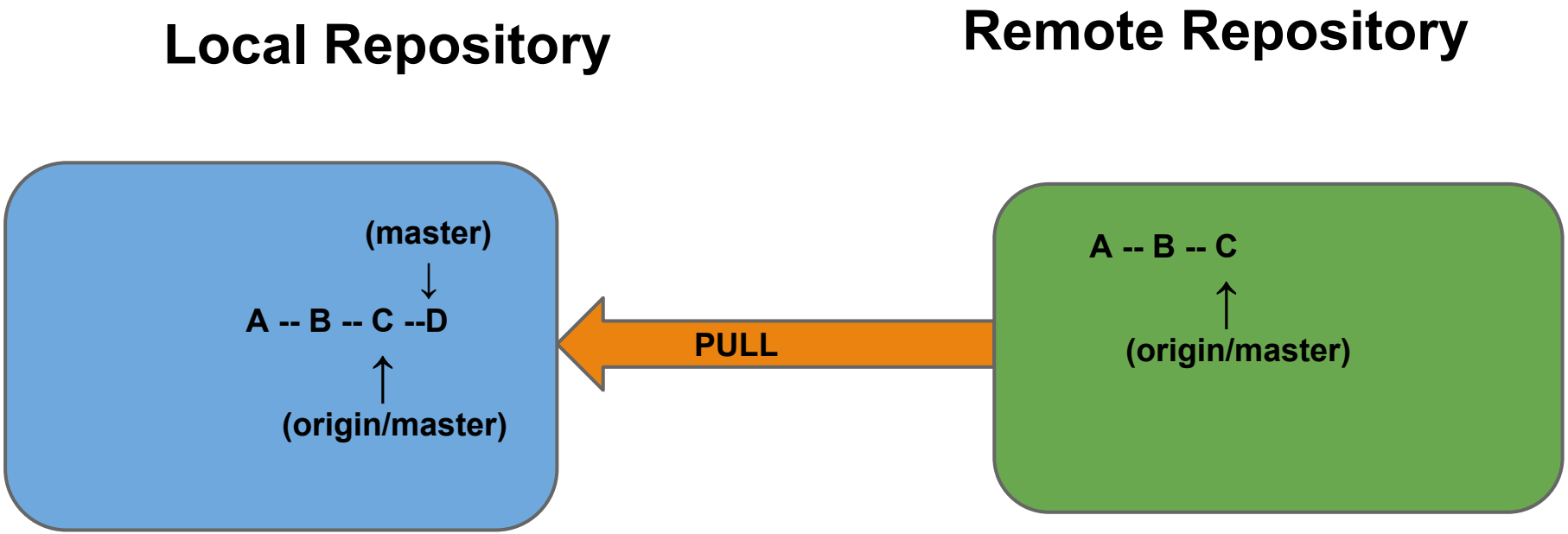


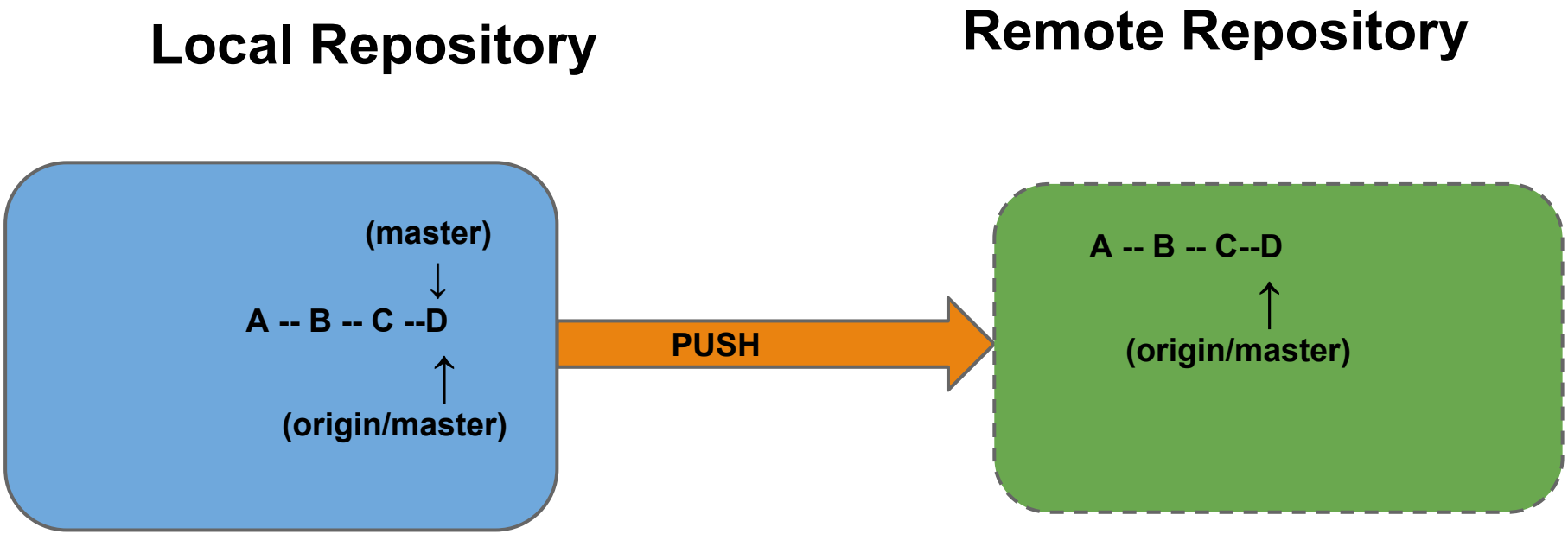
Remote Repository




Local Repository
Remote Repository
A -- B -- C
(origin/master)
PULL
A -- B -- C
(origin/master)

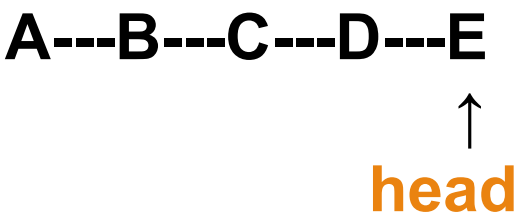

Local Repository
Remote Repository
(master)
A -- B -- C
(origin/master)
PULL
A -- B -- C
(origin/master)


Local Repository
Remote Repository
(master)
A -- B -- C --D
(origin/master)
PULL
A -- B -- C
(origin/master)

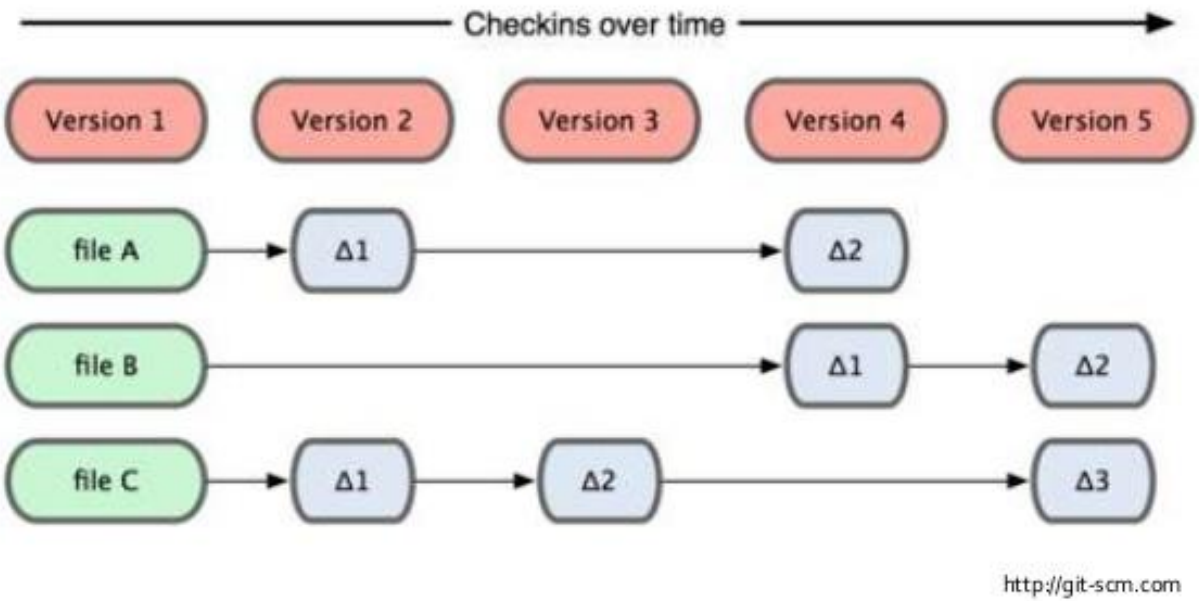

Local Repository
Remote Repository
(master)
A -- B -- C --D
(origin/master)
PUSH
A -- B -- C--D
(origin/master)

HEAD

It is the active commit that will be the parent of the next commit

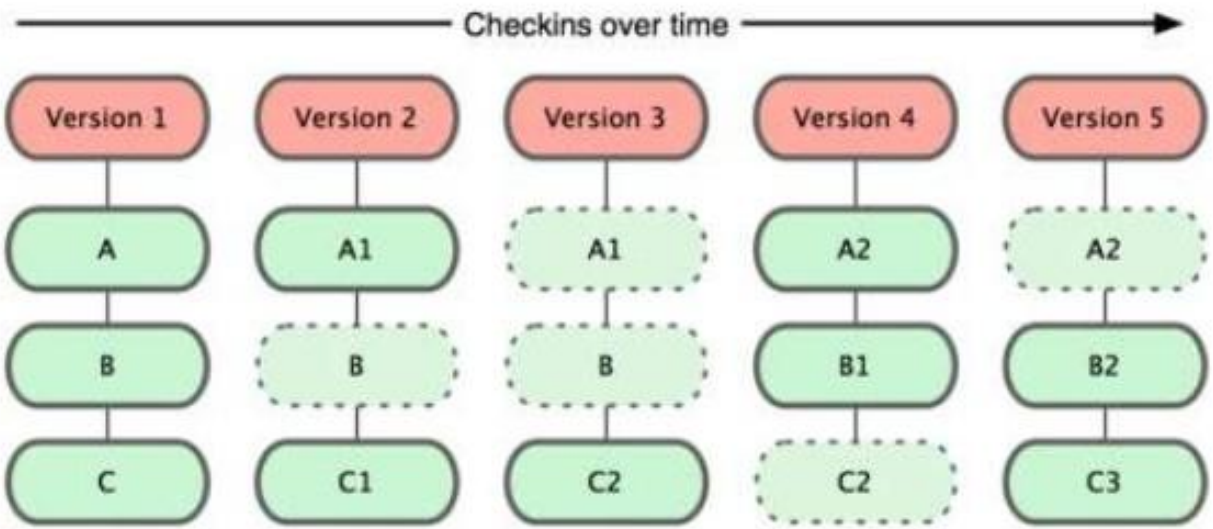


Snapshots, Not Differences



Differences

Snapshots, Not Differences



http://git-scm.com

Snapshots

Git Installation

Command Promt : git

Command Promt : gitk

Github Account

Go to www.github.com and register yourself there.

Github

Authentication

HTTP | SSH

SSH Key Generation and Setup

https://help.github.com/articles/generating-ssh-keys#platform-linux

https://help.github.com/articles/generating-ssh-keys#platform-windows

https://help.github.com/articles/generating-ssh-keys#platform-mac

Session 1

Exercise 1

Session 2

60 minutes


Git Commands

Git config

The git config command lets you configure your Git installation (or an individual repository) from the command line.

```
git config --global user.name <name>
git config --global user.email <email>
git config --global color.ui auto
```

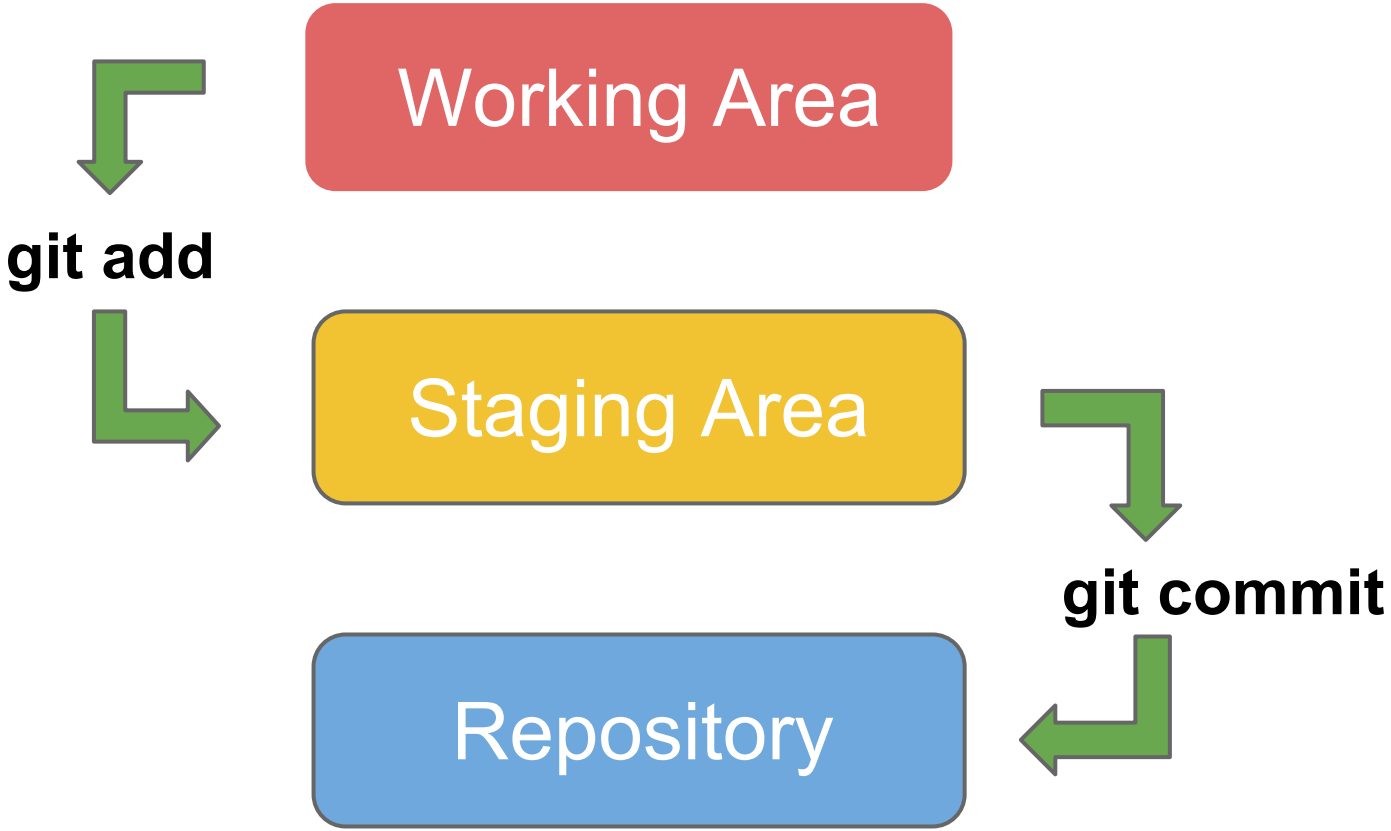
```
git config <parameter>
git config --list
```

Git init

To create a repository from an existing directory of files, you can simply run git init in that directory

Git status

List which files are staged, unstaged, and untracked.

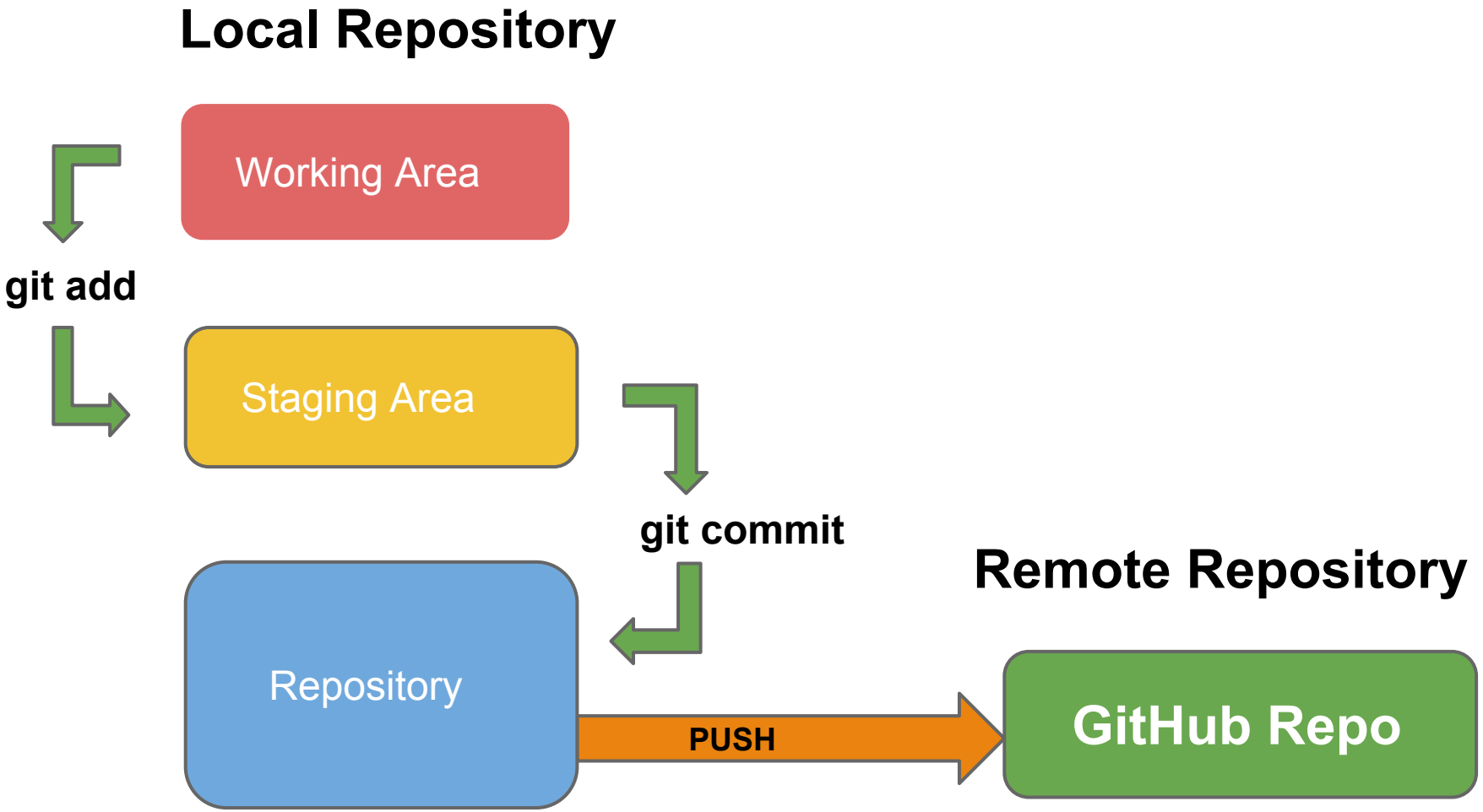

Working Area
git add
Staging Area
git commit
Repository

Git add

The git add command adds a change in the working directory to the staging area

Git commit

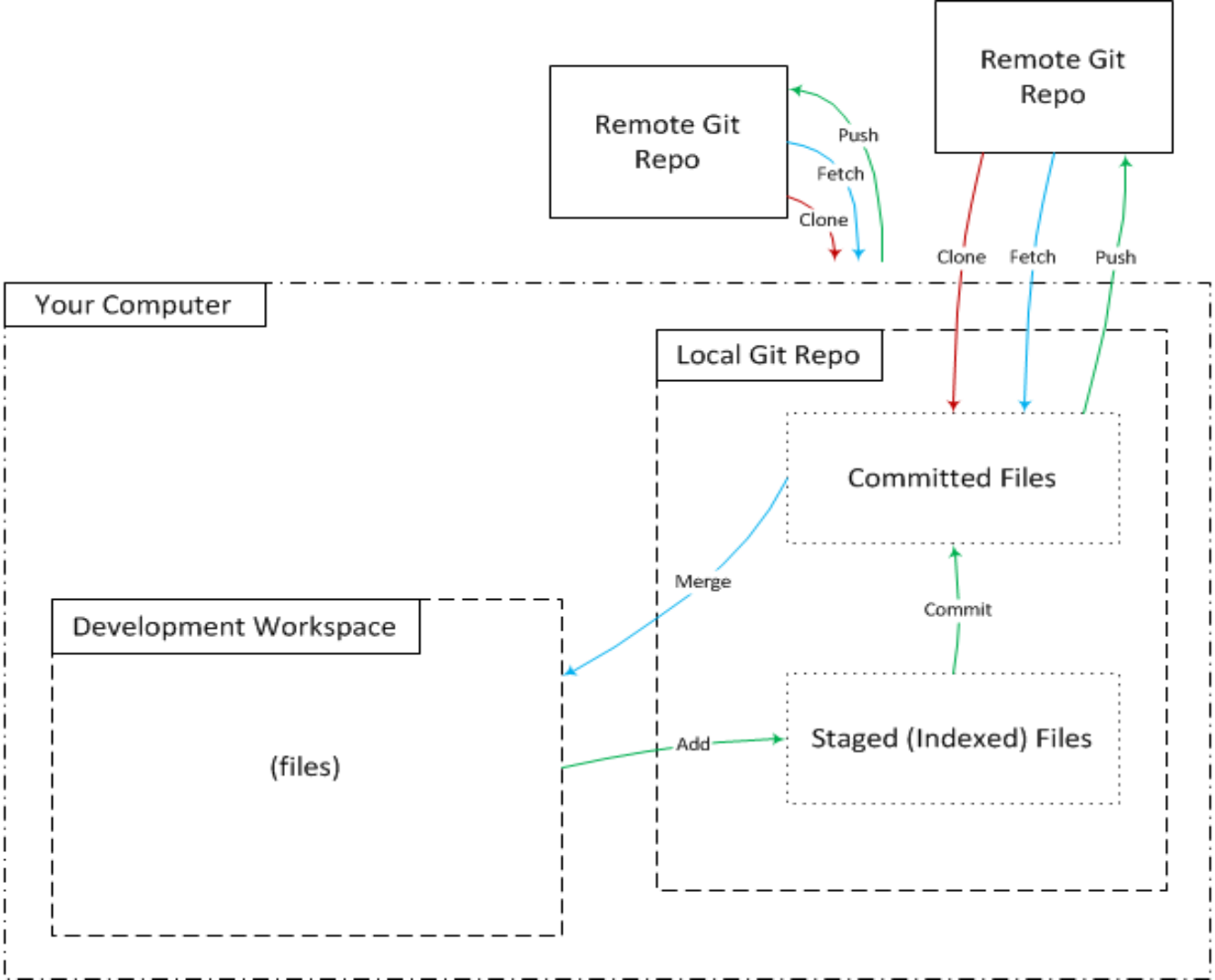
The git commit command commits the staged snapshot to the repository


Local Repository
Working Area
git add
Staging Area
git commit
Repository
Remote Repository
PUSH
GitHub Repo

Git push

Updates remote refs using local refs, while sending objects necessary to complete the given refs.

Git push <remote> <branch>


Remote Git Repo
Remote Git Repo
Push
Fetch
Clone
Clone
Fetch
Push
Your Computer
Local Git Repo
Committed Files
Merge
Commit
Development Workspace
(files)
Add
Staged (Indexed) Files

Session 2

Exercise 1

Undo Changes

Git checkout

Discard changes in working directory

checkout <file>

checkout <commit>

checkout <commit> <file>

Git reset <file>

Remove the specified file from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.

git reset --hard

Reset the staging area and the working directory to match the most recent commit. In addition to unstaging changes, the --hard flag tells Git to overwrite all changes in the working directory, too

Git revert

The git revert command undoes a committed snapshot. But, instead of removing the commit from the project history, it figures out how to undo the changes introduced by the commit and appends a new commit with the resulting content. This prevents Git from losing history, which is important for the integrity of your revision history and for reliable collaboration

Before the Revert

After the Revert

Reverting

Resetting

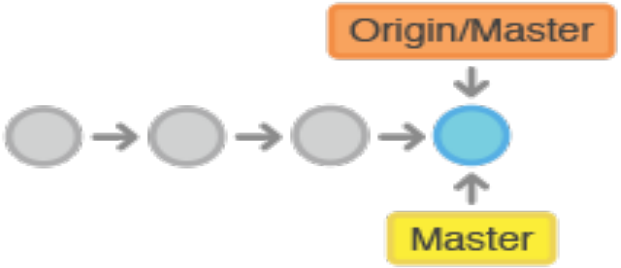
Git ignore

Ignoring the files that shouldn’t be tracked

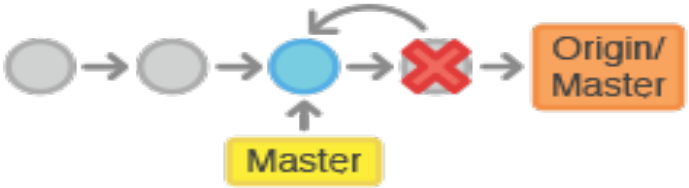
Session 2

Exercise 2

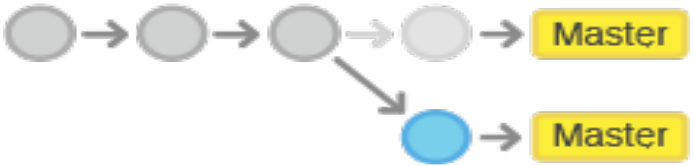
Before Resetting



After Resetting



After Committing

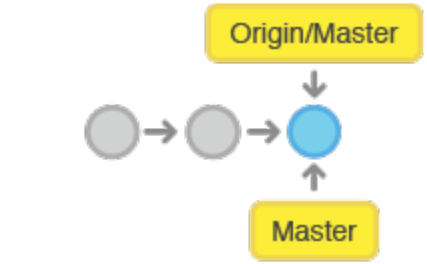


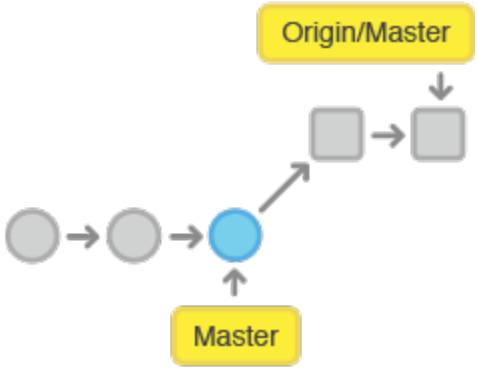
Git clone

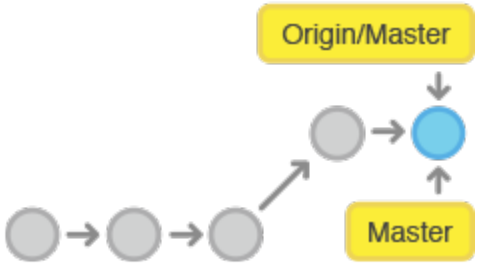
The git clone command copies an existing Git repository. This is sort of like svn checkout, except the “working copy” is a full-fledged Git repository—it has its own history, manages its own files, and is a completely isolated environment from the original repository

Git fetch

The git fetch command imports commits from a remote repository into your local repo. The resulting commits are stored as remote branches instead of the normal local branches that we've been working with. This gives you a chance to review changes before integrating them into your copy of the project.


Origin/Master
Master


Origin/Master
Master


Origin/Master
Master

git fetch origin

git fetch origin <branch>

Fetch all of the branches from the repository. This also downloads all of the required commits and files from the other repository

Git pull

Fetch the specified remote's copy of the current branch and immediately merge it into the local copy. This is the same as git fetch <remote> followed by git merge origin/<current-branch>.

```
git pull origin
git pull origin <branch>
```

Scenarios

- When you have **no local changes**
- When you have **new local files** in **unstaged area**
- When you have **new local files** in **staged area**
- When you have **existing modified files** in **unstaged area**
- When you have **existing modified files** in **staged area**
- When you have **local commits**
- **Mix of everything above**

No Problem Until Conflicting Changes

Git stash

Stashing takes the dirty state of your working directory — that is, your modified tracked files and staged changes — and saves it on a stack of unfinished changes that you can reapply at any time

Session 2

Exercise 3

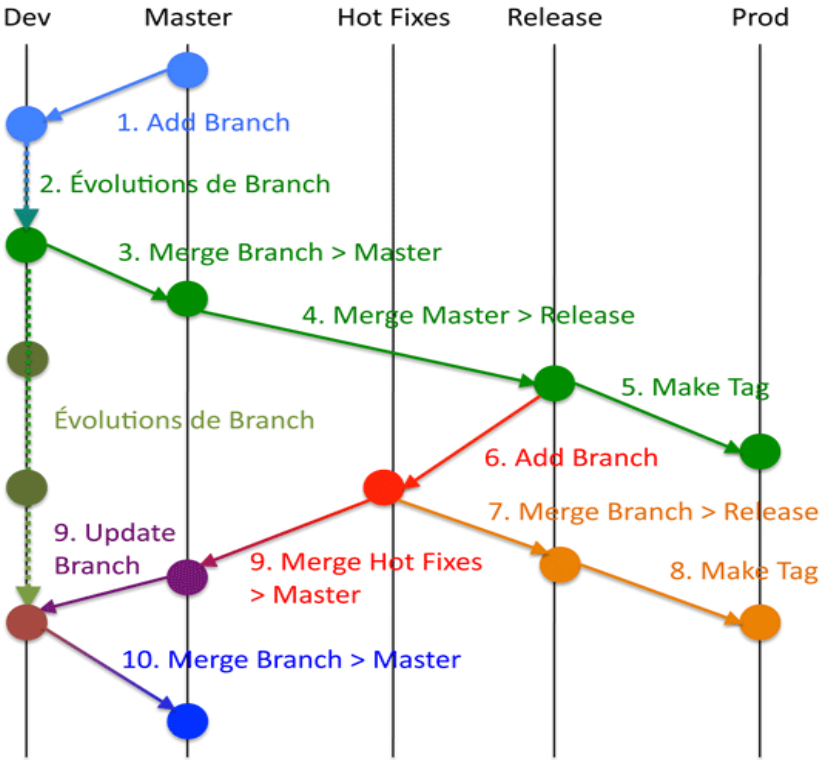
End Session 2!

Session 3

60 minutes

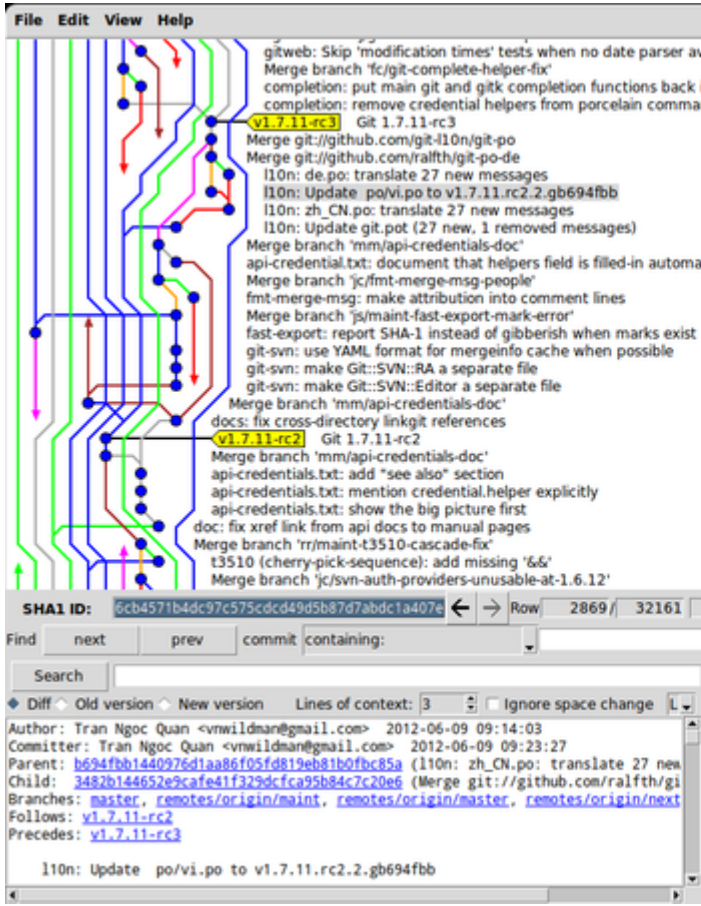
Branch & Merge



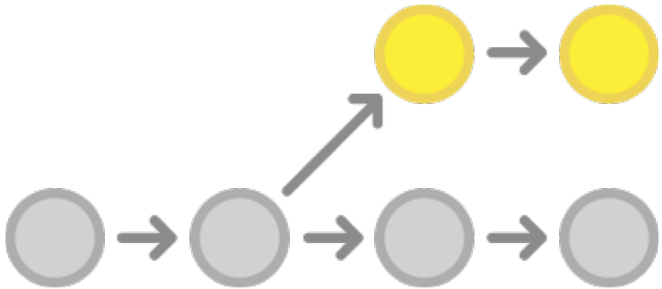
Git K

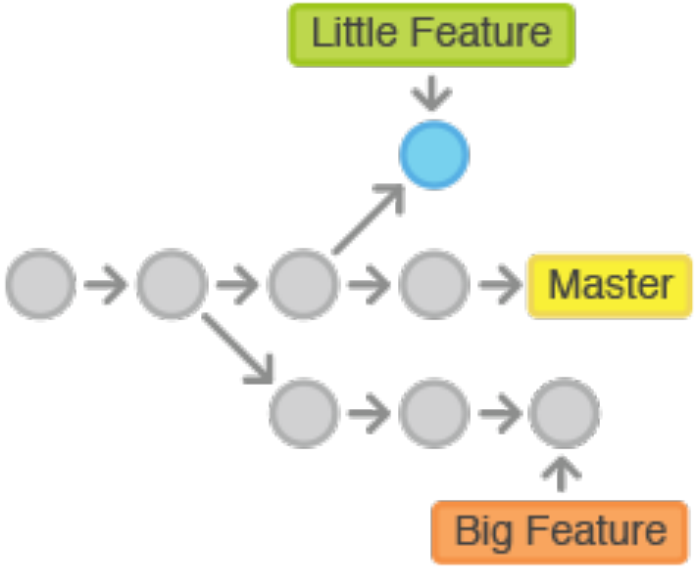
Displays changes in a repository or a selected set of commits. This includes visualizing the commit graph, showing information related to each commit, and the files in the trees of each revision.



Git branch

A branch represents an independent line of development. Branches serve as an abstraction for the edit/stage/commit process




Little Feature
Master
Big Feature

git branch - Show current branch

git branch <branch> - Create a new branch

git checkout <branch> - Move to branch

git push origin <branch> - Push branch to origin

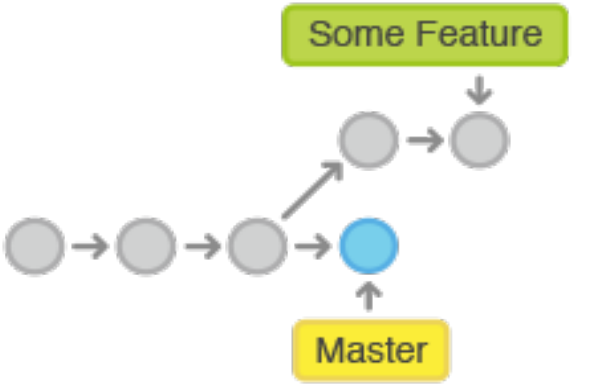
git branch -d <branch> - Delete branch locally

git push origin --delete <branchName> - Delete branch from origin

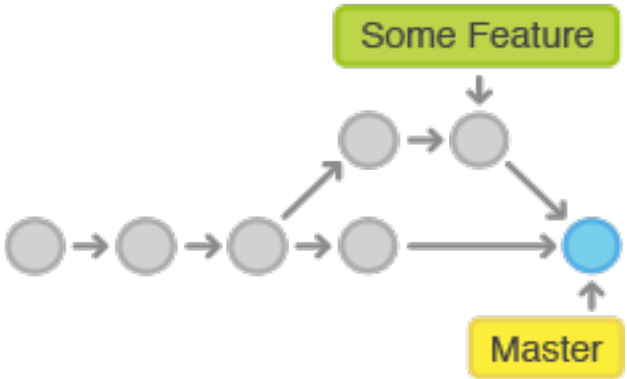
Git merge

Merging is Git's way of putting a forked history back together again. The git merge command lets you take the independent lines of development created by git branch and integrate them into a single branch.

Before Merging



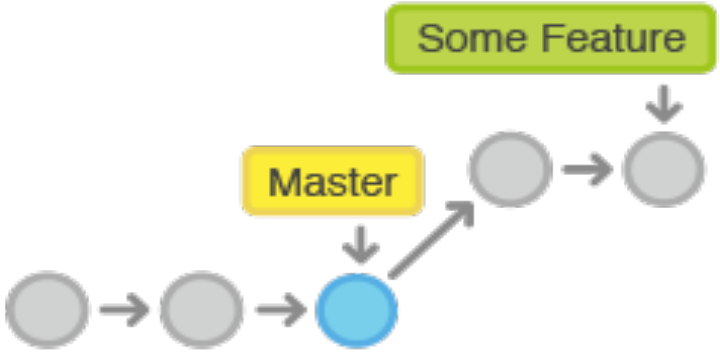
After a 3-way Merge



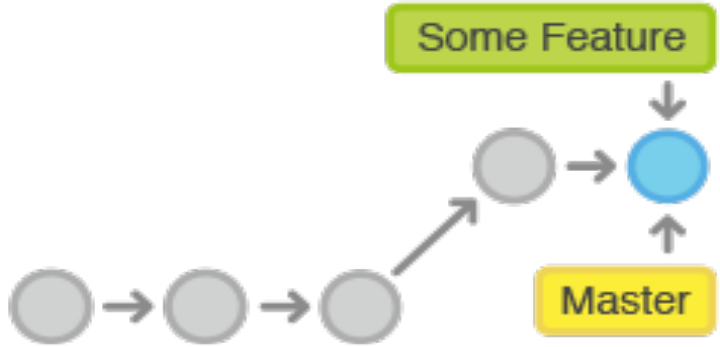
git merge <branch>

Fast Forward Merge

Before Merging



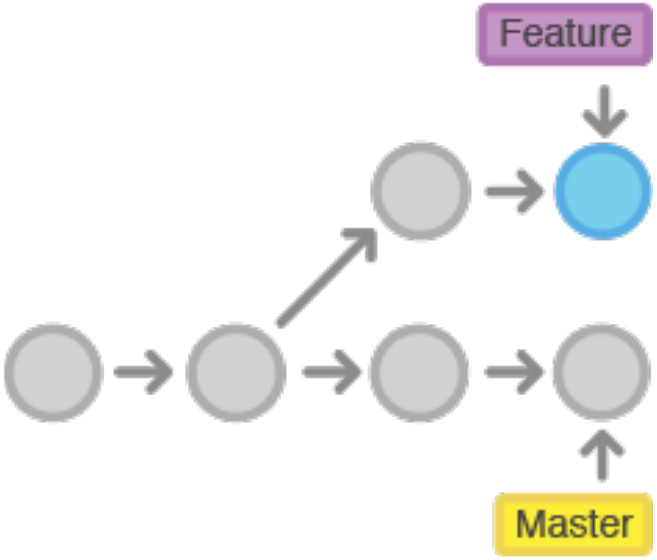
After a Fast-Forward Merge



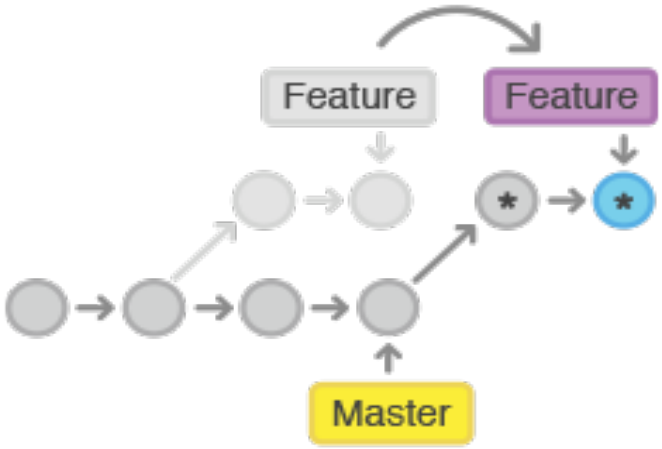
git merge --no-ff <branch>

Git rebase

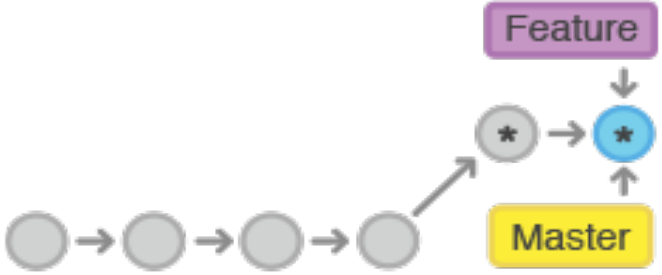
Rebasing is the process of moving a branch to a new base commit. The general process can be visualized as the following



After Rebasing Onto Master



After a Fast-Forward Merge



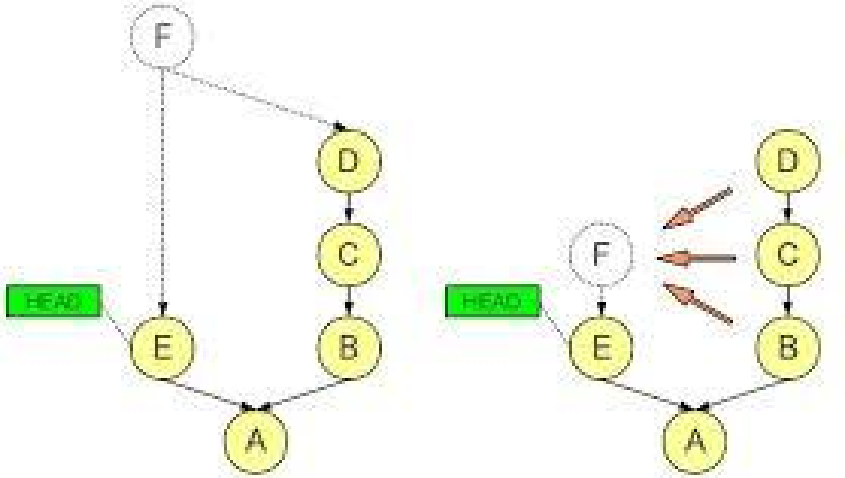
*Brand New Commits

```
git rebase <base>
git checkout <base>
git merge <feature>

git push origin <base>
git push origin <feature>
```

- For Short Lived Branches
- No Rebase for Public Commits

Git squash


F
D
C
HEAD
E
B
A
D
F
C
HEAD
E
B
A

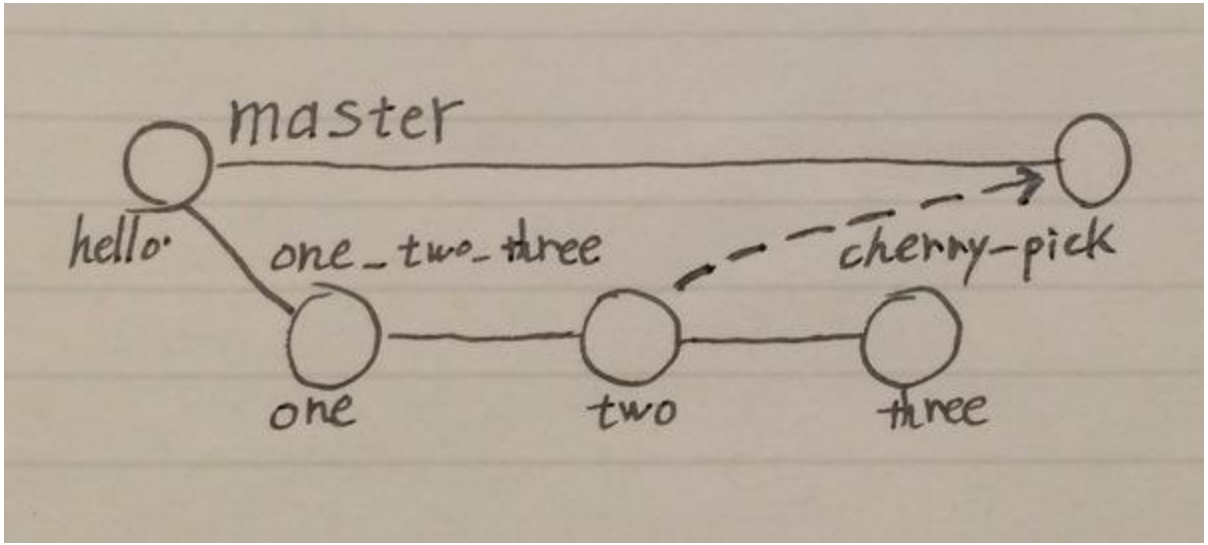
git rebase -i <base>

Session 3

Exercise 1

Git cherry

Cherry picks <commits>/<tags>/<others>


master
hello
one_two_three
cherry-pick
one
two
three

git cherry-pick <commit>

Git log

Show commit logs

Git show

Show details about <commit>/<tag>/<others>

Git grep

Lets you search through your trees of content for words and phrases

Git mv

renaming a file

Git diff

Show the difference between files

Git tag

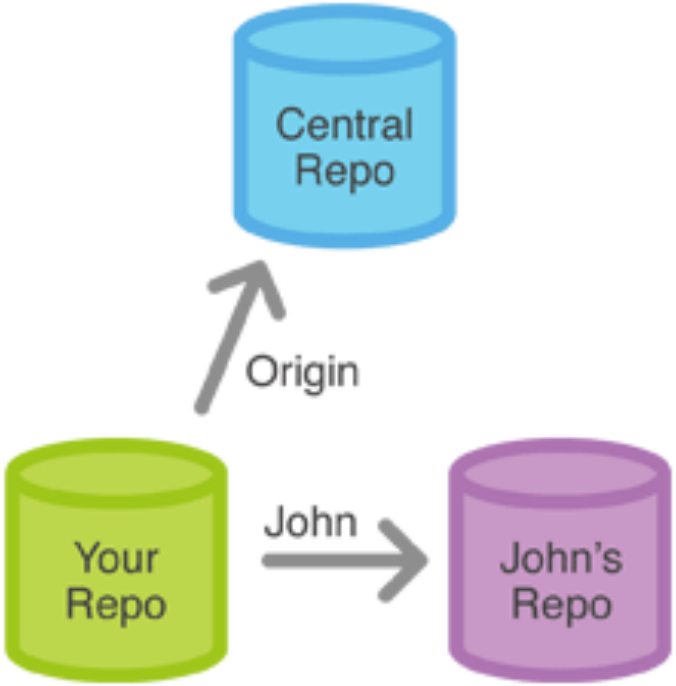
Like most VCSs, Git has the ability to tag specific points in history as being important

git tag -a v1.0 -m '1.0 Version'

git tag - List all tags

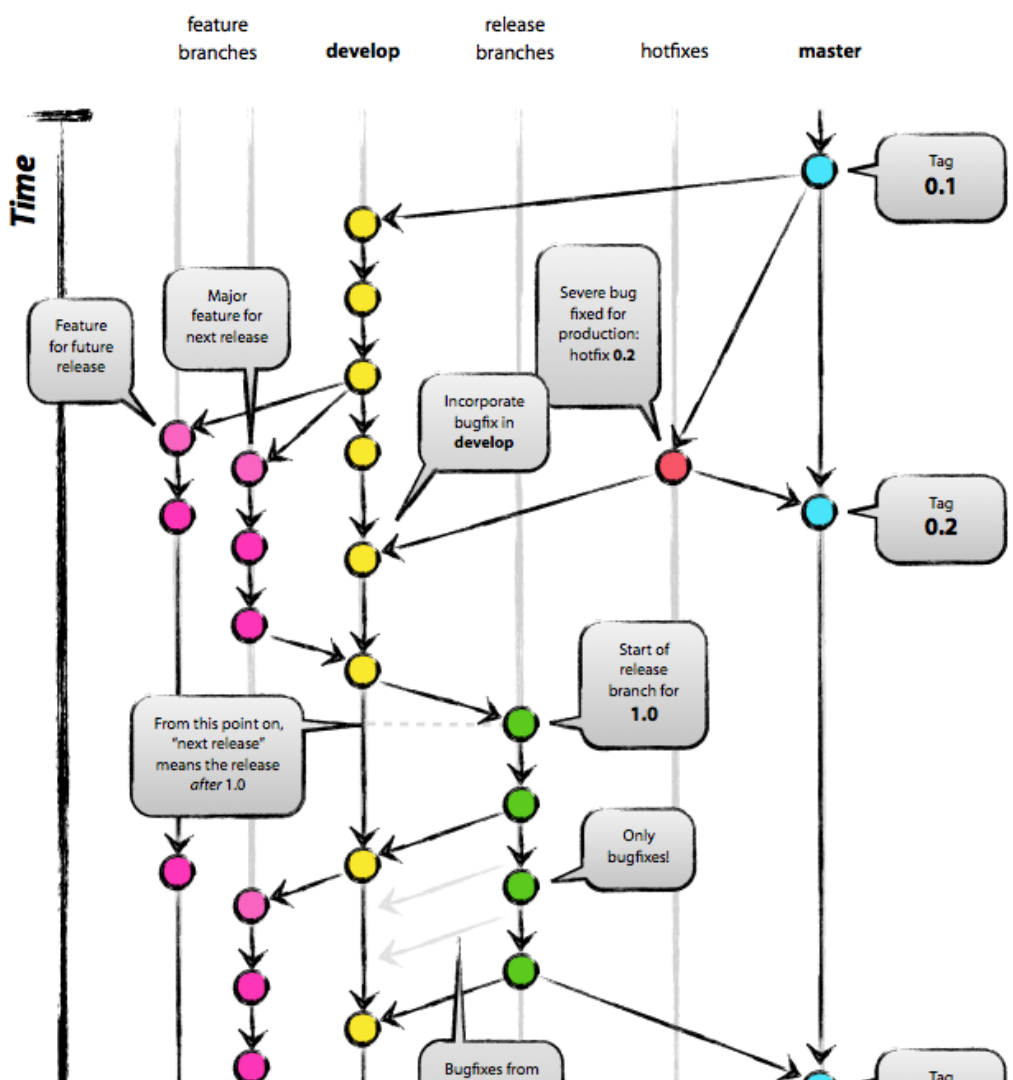
Git remote

The git remote command lets you create, view, and delete connections to other repositories


Central
Repo
Origin
Your
Repo
John
John's
Repo

```
git remote
git remote -v
git remote add <name> <url>
git remote rm <name>
```

Git in Projects


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from

Pull Request

Pull requests let you tell others about changes you've pushed to a GitHub repository. Once a pull request is sent, interested parties can review the set of changes, discuss potential modifications, and even push follow-up commits if necessary

Session 3

Exercise 2

End!