# GITHub Essentials

The basics of GITHub

# Overview of GIT

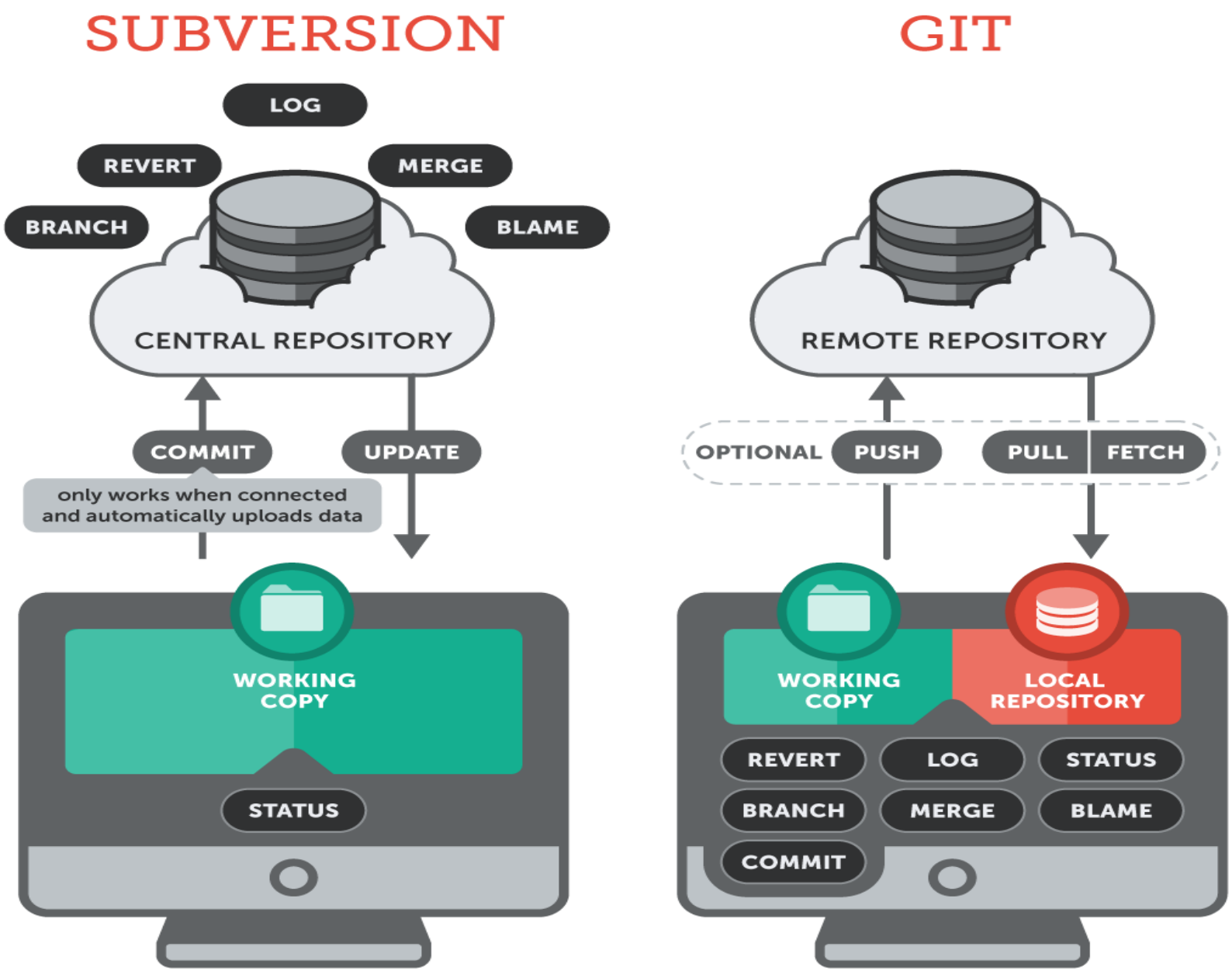
# Overview

- Git is a **distributed revision control** and source code management (SCM) system
- Emphasis on **speed, data integrity** and support for distributed, non-linear workflows.
- Every Git working directory is a full-fledged repository with complete history and full version-tracking capabilities, independent of network access or a central server.
- Like the Linux kernel, **Git is free software** distributed under the terms of the GNU General Public License version 2.

# Design Base

- Git's design was inspired by **BitKeeper and Monotone.**
- Git was originally designed as a low-level version control system engine on top of which others could write front ends, such as **Cogito** .
- The core Git project has since become a complete version control system that is usable directly.
- While strongly influenced by BitKeeper, Torvalds deliberately attempted to avoid conventional approaches, leading to a unique design.

# CVC vs DVC


SUBVERSION
LOG
REVERT
MERGE
BRANCH
BLAME
CENTRAL REPOSITORY
COMMIT
UPDATE
only works when connected
and automatically uploads data
WORKING
COPY
STATUS
GIT
REMOTE REPOSITORY
OPTIONAL
PUSH
PULL
FETCH
WORKING
COPY
LOCAL
REPOSITORY
REVERT
LOG
STATUS
BRANCH
MERGE
BLAME
COMMIT

# Characteristics

- **Strong support for non-linear development**
- **Distributed development**
- **Compatibility with existing systems/protocols**
- **Efficient handling of large projects**
- **Cryptographic authentication of history**
- **Toolkit-based design**
- **Pluggable merge strategies**
- **Garbage accumulates unless collected**
- **Periodic explicit object packing**

# GIT server

## Gitolite

- Gitolite is an access control layer on top of git, providing fine access control to git repositories. It relies on other software to remotely view the repositories on the server.

## Gerrit

- Gerrit provides two out of three functionalities: access control, and managing repositories. It uses jGit. To view repositories it is combined e.g. with Gitiles or GitBlit.

## Gitblit

- Gitblit can provide all three functions, but is in larger installations used as repository browser installed with gerrit for access control and management of repositories.

## Gitiles

- Gitiles is a simple repository browser, usually used together with gerrit.

## Bonobo Git Server

- Bonobo Git Server is a simple git server for Windows implemented as an ASP.NET gateway.It relies on the authentication mechanisms provided by Windows Internet Information Services, thus it does not support SSH access but can be easily integrated with Active Directory.

## Commercial solutions

- Commercial solutions are also available to be installed on premises, amongst them GitHub Software (using native git, available as a vm),Stash (using jGit), Team Foundation Server (using libgit2).

# Basics GIT commands

# Basic Terms

## Non Bare Repository

- A bare repository in Git is a normal repository which can be used to version control files by executing git commit commands
- This repository is initialized with a .git folder

## Bare Repository

- A central repository can be created as non bare repository and multiple developers are working on this repository and pushing their changes (won't be allowed but assume).
- There will not be a git commit in this repository

## Branch

- Branch is essentially an independent line of development that enables you to isolate your work from others.

# Characteristics

# Git Cheat Sheet

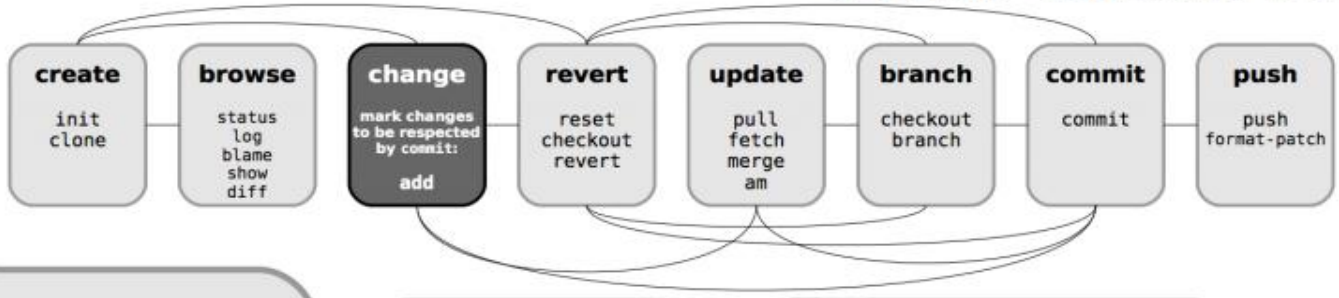
by Jan Krüger <jk@jk.gs>, http://jan-krueger.net/git/
Based on work by Zack Rusin

## Basics

Use git help [*command*] if you're stuck.

| | |
|---|---|
| `master` | default devel branch |
| `origin` | default upstream branch |
| `HEAD` | current branch |
| `HEAD^` | parent of HEAD |
| `HEAD~4` | great-great grandparent of HEAD |
| *`foo..bar`* | from branch *foo* to branch *bar* |

## Command Flow (left to right)




## Create

**From existing files**
```
git init
git add .
```
**From existing repository**
```
git clone ~/old ~/new
git clone git://...
git clone ssh://...
```

## View

```
git status
git diff [oldid newid]
git log [-p] [file|dir]
git blame file
git show id (meta data + diff)
git show id:file
git branch (shows list, * = current)
git tag -l (shows list)
```

## Revert

In Git, revert usually describes a new commit that undoes previous commits.

```
git reset --hard (NO UNDO)
        (reset to last commit)
git revert branch
git commit -a --amend
        (replaces prev. commit)
git checkout id file
```

## Publish

In Git, `commit` only respects changes that have been marked explicitly with `add`.

```
git commit [-a]
        (-a: add changed files automatically)
git format-patch origin
        (create set of diffs)
git push remote
        (push to origin or remote)
git tag foo
        (mark current version)
```

## Update

```
git fetch (from def. upstream)
git fetch remote
git pull (= fetch & merge)
git am -3 patch.mbox
git apply patch.diff
```

## Branch

```
git checkout branch
        (switch working dir to branch)
git merge branch
        (merge into current)
git branch branch
        (branch current)
git checkout -b new other
        (branch new from other and switch to it)
```

## Useful Tools

```
git archive
        Create release tarball
git bisect
        Binary search for defects
git cherry-pick
        Take single commit from elsewhere
git fsck
        Check tree
git gc
        Compress metadata (performance)
git rebase
        Forward-port local changes to remote branch
git remote add URL
        Register a new remote repository for this tree
git stash
        Temporarily set aside changes
git tag
        (there's more to it)
gitk
        Tk GUI for Git
```

## Conflicts

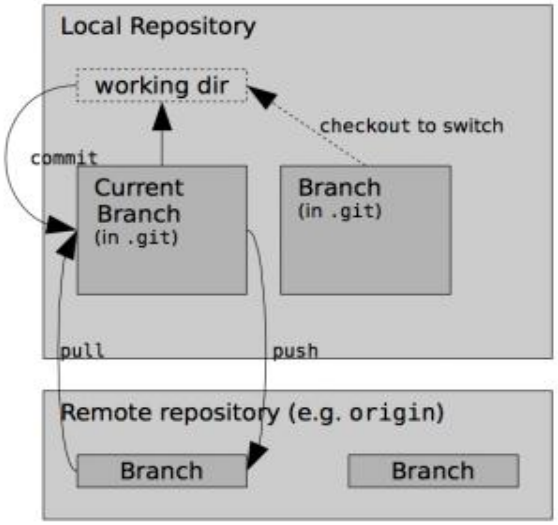
Use add to mark files as resolved.

```
git diff [--base]
git diff --ours
git diff --theirs
git log --merge
gitk --merge
```

## Tracking Files

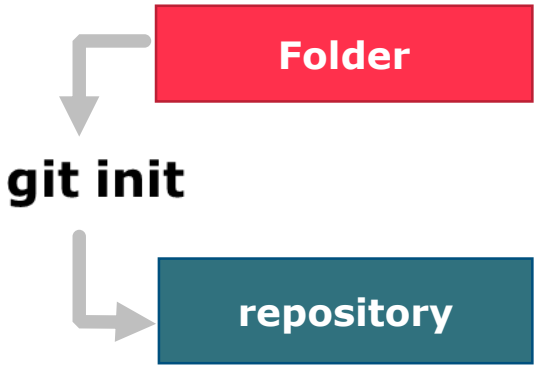
```
git add files
git mv old new
git rm files
git rm --cached files
        (stop tracking but keep files in working dir)
```

## Structure Overview

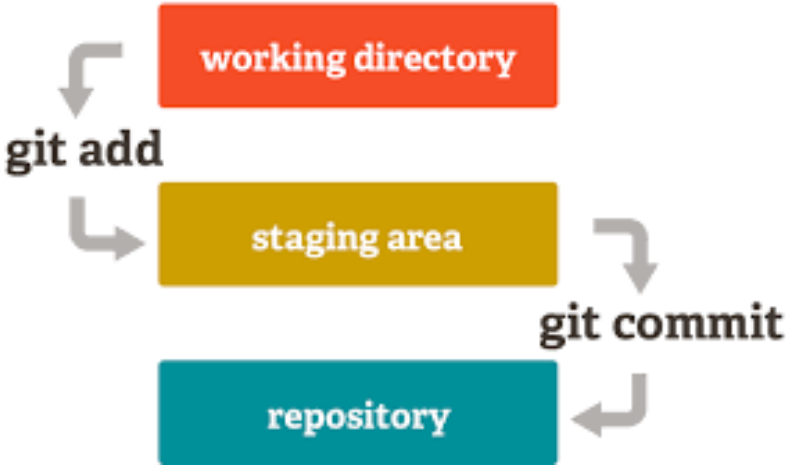


# Basic GIT Commands

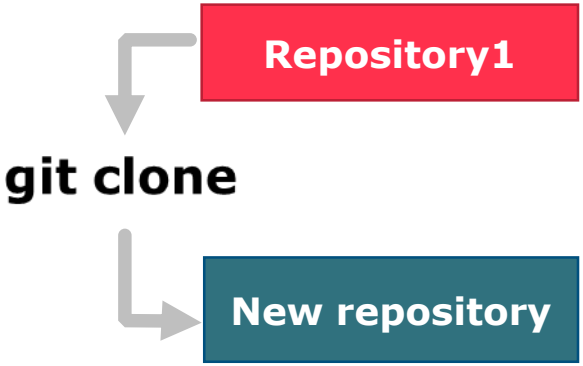
## git init




## git commit

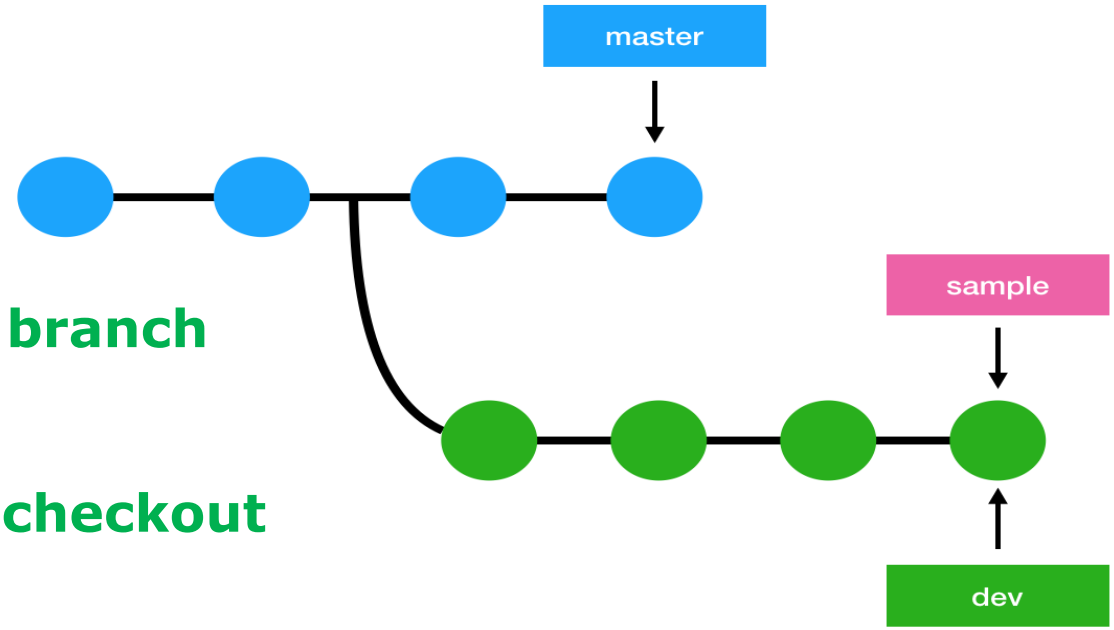



## git clone

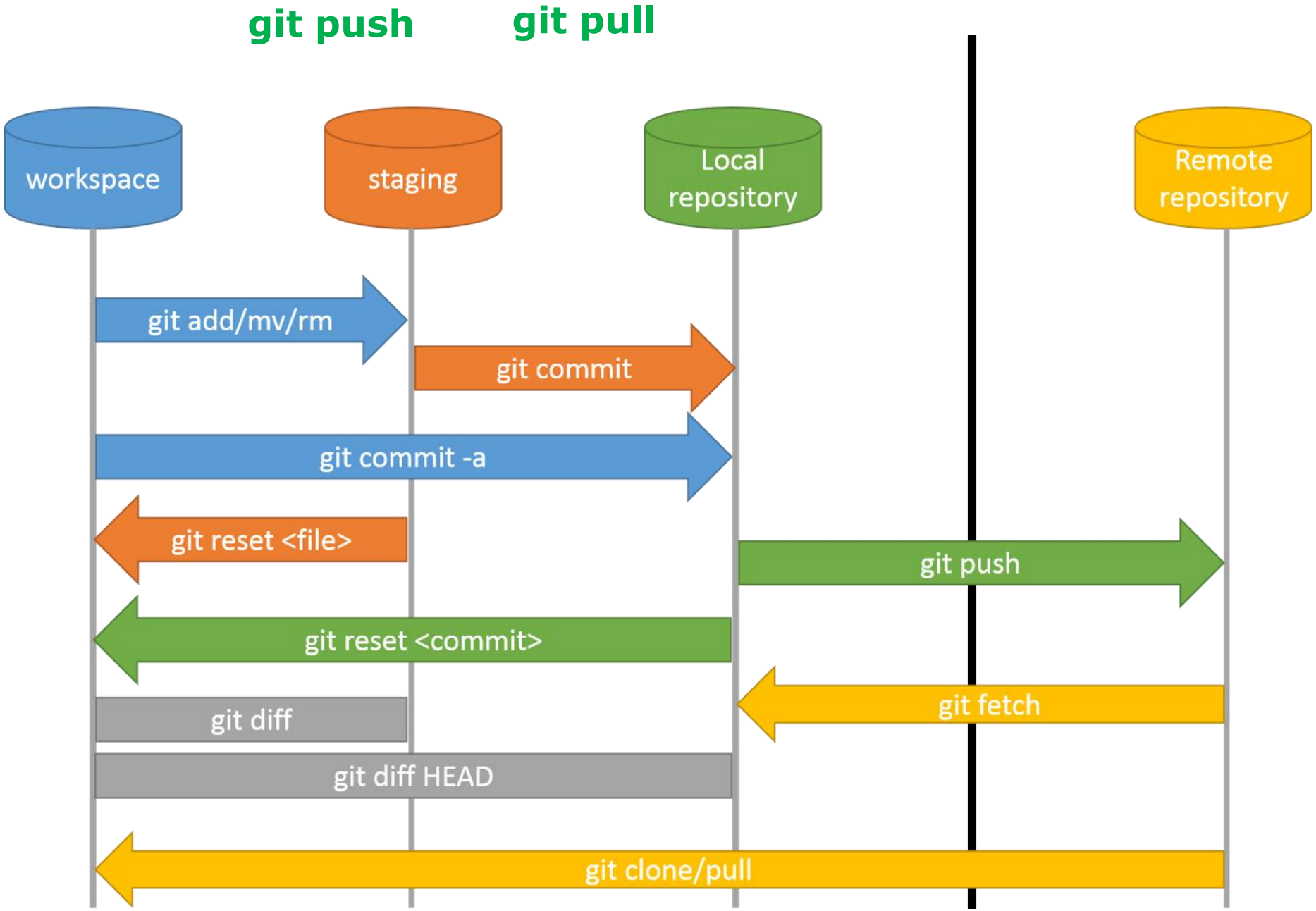



## git branch

## git checkout



# Basic GIT Commands




# Demo on GIT commands

# Introduction to GitHub

# GitHub Introduction

- **GitHub** is a web-based hosting service for version control using git.
- It is mostly used for computer code. It offers all of the distributed version control and source code management(SCM) functionality of Git as well as adding its own features.
- It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.
- GitHub offers plans for both private repositories and free accounts which are commonly used to host open-source software projects

Trusted by more than 1.8M* businesses and organizations

airbnb IBM SAP PayPal Spotify Bloomberg

* As of March 2018

# Git Vs GitHub

## Git vs. GitHub


| Git | GitHub |
|---|---|
| a tool that allows creating a local repository | allows hosting the central repository on a remote server |
| a version control **system** | a repository hosting **service** |

# GitHub Features

- git hosting provider server
- easy contributions of code between our project members
- collaboration features
  - read me, wiki
  - pull requests
  - commit history
  - access control to various collaborators.
  - issue tracking
- feature to compare two branches or two commits
- Easy to use Web Based Graphical interface

# GitHub Hosting Server

- GitHub provides unlimited public and private repositories
- Users can created Repositories and store code
- Commit, Branching and Pull within repositories
- Local repositories can be pushed to remote repositories


Code hosting

## All your code in one place

GitHub is one of the largest code hosts in the world with over 80 million* projects. Private, public, or open source, all repositories are equipped with tools to help you host, version, and release code.

* As of March 2018

# GitHub Hosting Server

**GitHub Repositories**

- Users can create Repositories in GitHub
- Repositories are classified based on Access as
  - ❖ Public repositories
    - ✓ Credentials are required to create and store code in Public repositories
    - ✓ Any one can download the content in a public repository
  - ❖ Private repositories
    - ✓ These are available only for a paid account
    - ✓ Access to the content of the repository is controlled by the creator of the repository

# GitHub Hosting Server

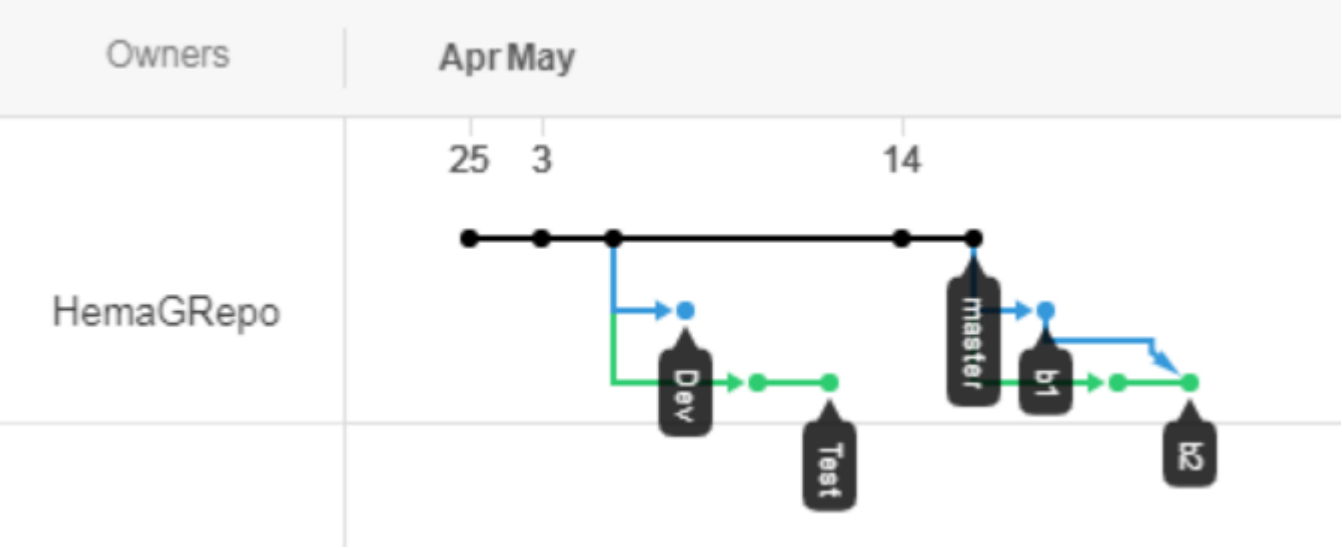
**GitHub Repositories**

- Repositories are classified based on functionality as
  - ❖ Git repositories
    - ✓ Similar to creating repositories with git init
    - ✓ Commits can happen in these repositories
  - ❖ Git bare repositories
    - ✓ These are central repositories
    - ✓ Commit cannot be done on a bare repository
    - ✓ They are targets of git push and they act as a central place to store code from multiple developers

# GitHub Hosting Server

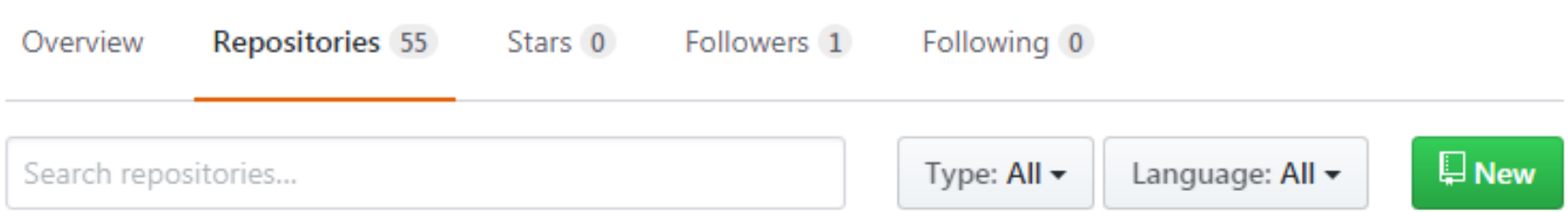
## GitHub Branches

- A git repository has a master branch on which all comments happen
- New Branches can be created and commits can happen in different branches
- GitHub provides a branch history to view the HEAD of each commit

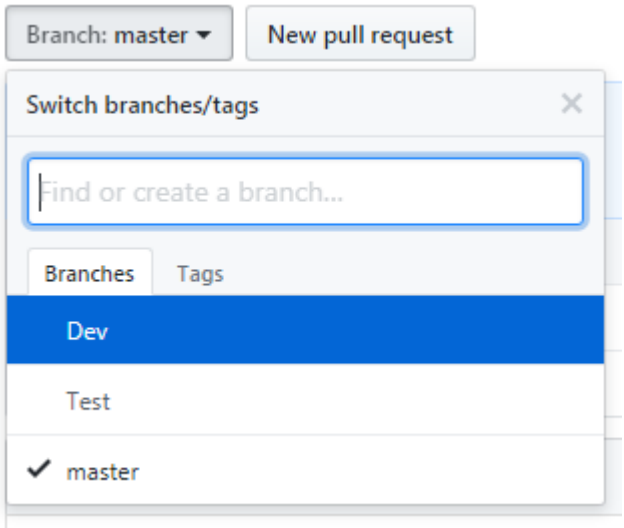


# Demo


## Creating Repositories in GitHub




## Creating Branches in Repositories in GitHub



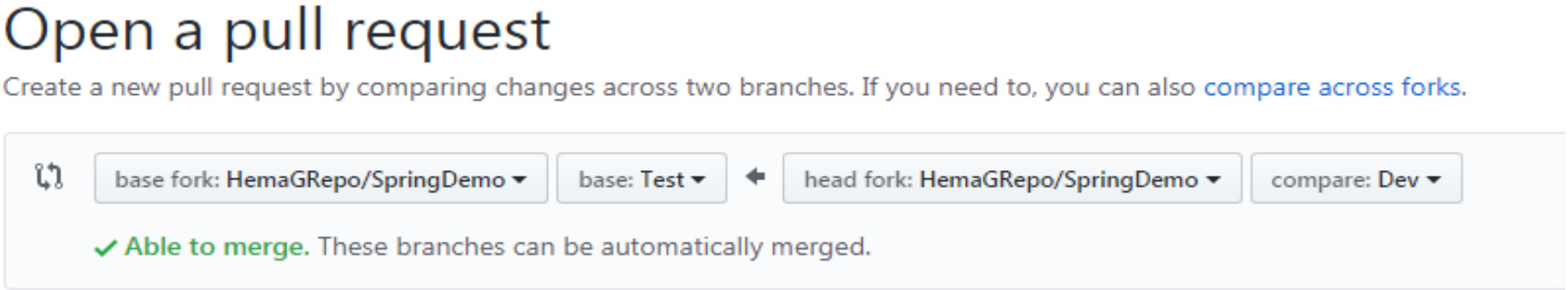
# GitHub Hosting Server

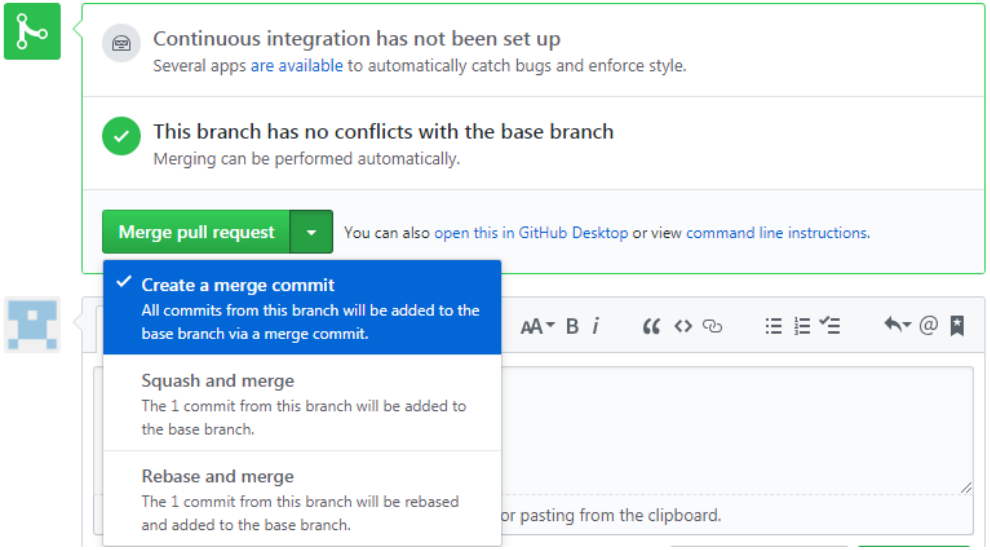
**GitHub Pull and Merge**

- Git hub provides options to submit pull requests
- Pull requests let you tell others about changes you've pushed to a repository on GitHub.
- Once a pull request is opened, once can review the potential changes before the changes are merged into the repository.
- Merge can be done on a pull request into the upstream branch when work is completed.
- Anyone with push access to the repository can complete the merge.
- If the changes in a topic branch need not be merged to the upstream branch, the pull request can be closed without merging.

# Demo


## Pull request for branches in GitHub



# Issue Tracking in GitHub

**Issue Tracking**

- Github provides options for Code review and issue tracking
- Initially an issue will be open state and the Issue tracking tab provides options for the following:
  - Deal with your issues just like you deal with email
  - Create and apply labels to issues to assign to users or categorize
  - Drag and drop issues to prioritize them
  - Search, sort, and filter
  - Close issues from commit messages
- An issue can be automatically closed when a commit references an issue number

# GITHub desktop

# GitHub Desktop

**What is GitHub Desktop**

- GitHub Desktop (formerly GitHub for Windows) is a more streamlined GUI.
- Uses PowerShell for git command line.
- The version of git it includes lags the latest release.
- "Before you set up GitHub Desktop, you must already have a GitHub or GitHub Enterprise account."
- You log in with your account, your GitHub repositories are automatically detected.

# GitHub Desktop

## Installing GitHub Desktop

- You can install GitHub Desktop on Microsoft Windows 7 or later .
- Visit the GitHub Desktop download page.
- Choose Download for Windows.
- In your computer's Downloads folder, double-click GitHub Desktop.
- In the pop-up window, click Install.
- After the program has been installed, click Run.

Add your GitHub.com or GitHub Enterprise account information to GitHub Desktop so you can access your repositories.

# GitHub Desktop

## Creating Repositories

- GitHub Desktop provides 3 options for creating repositories
  - ❖ Creating New Repositories
  - ❖ Add a local repository

## Creating New Repositories

- New Git repositories can be created using the Create New Repository Option
- All Git commands can be executed in the repository created

# GitHub Desktop

**Adding a Local repository**

- GitHub Desktop provides a GUI to monitor a local GIT repository
- Repository Explorer and Status view are available
- Changes to a local repository can be monitored and commits can be issued.
- Branching is also possible in GitHub desktop
- The branches can be pushed to the linked GitHub repository
- Automatically in remote repository the commit happens on the same branch
- Automatic detection of changes in the content of the repository


# Demo on GITHub Desktop