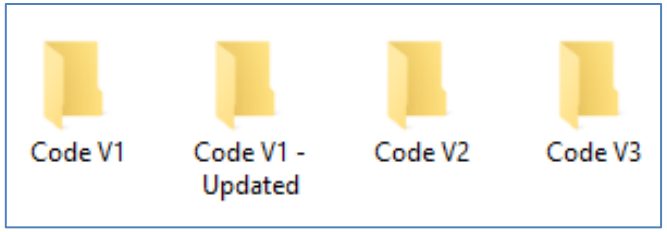
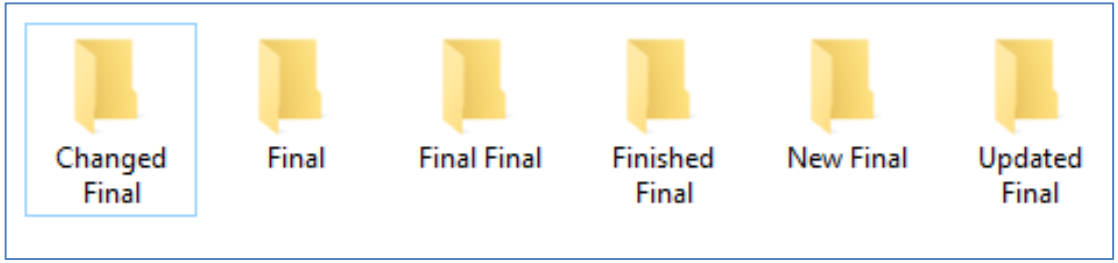
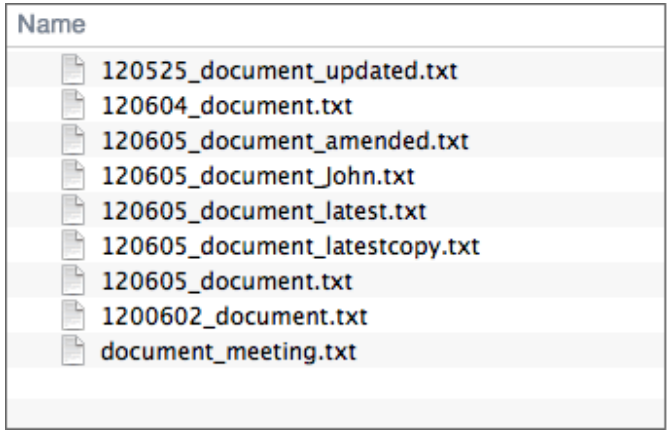
Open Source Software Development

Ung Văn Giàu
Email: giau.ung@eiu.edu.vn

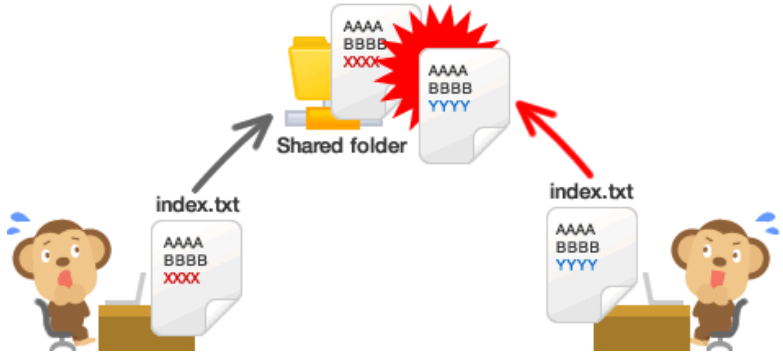
Version Control System


Code V1
Code V1 - Updated
Code V2
Code V3

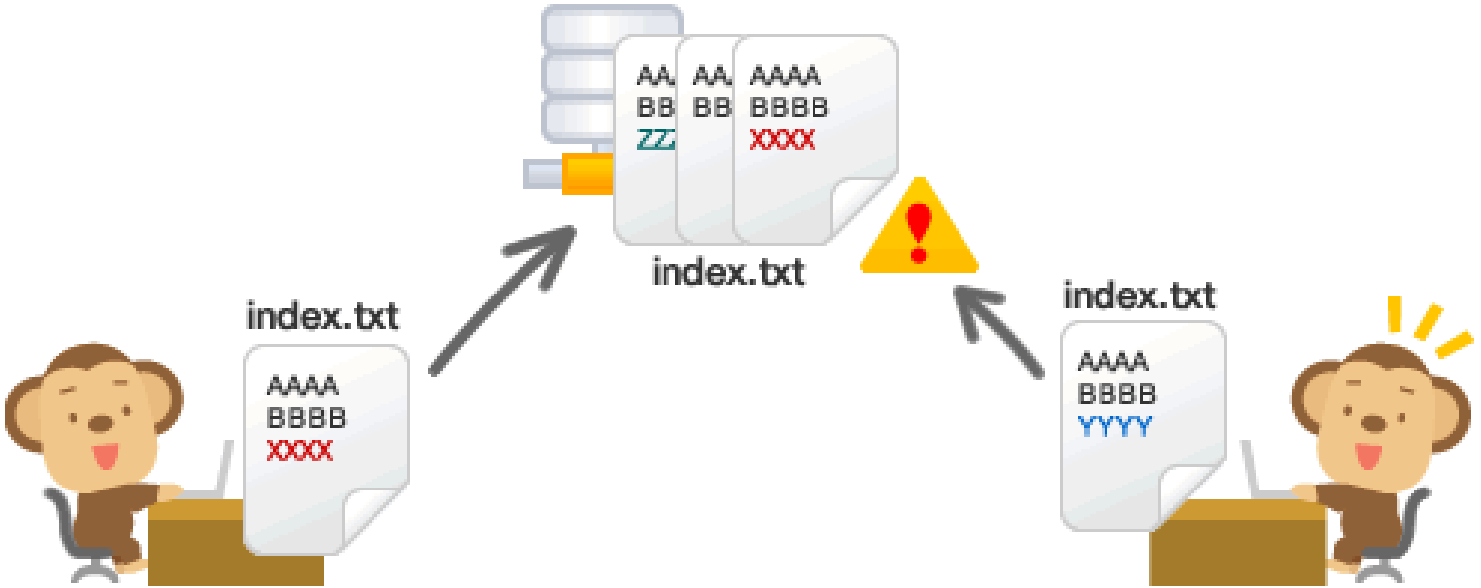

Changed Final
Final
Final Final
Finished Final
New Final
Updated Final


Name
120525_document_updated.txt
120604_document.txt
120605_document_amended.txt
120605_document_John.txt
120605_document_latest.txt
120605_document_latestcopy.txt
120605_document.txt
1200602_document.txt
document_meeting.txt

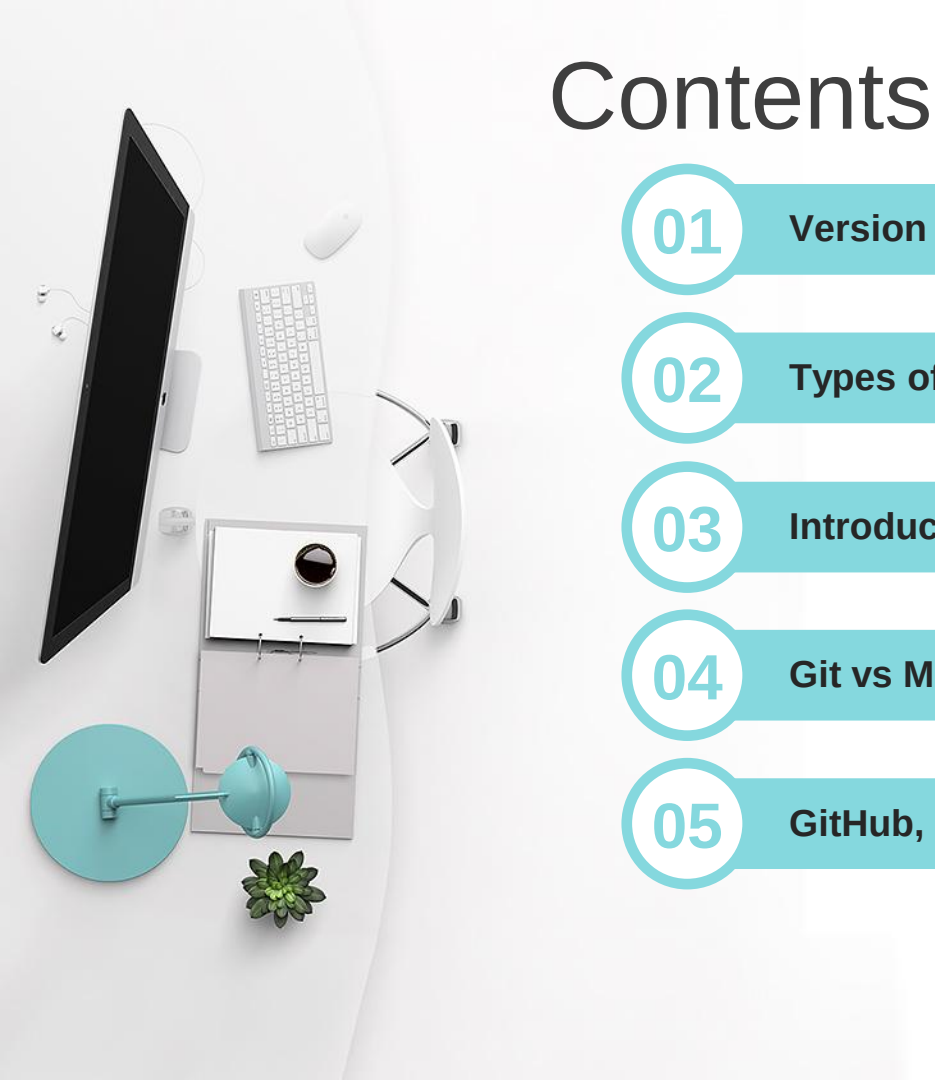

Nhìn cái này chỉ muốn khóc thôi...


AAAA
BBBB
XXXX
AAAA
BBBB
YYYY
Shared folder
index.txt
AAAA
BBBB
XXXX
index.txt
AAAA
BBBB
YYYY

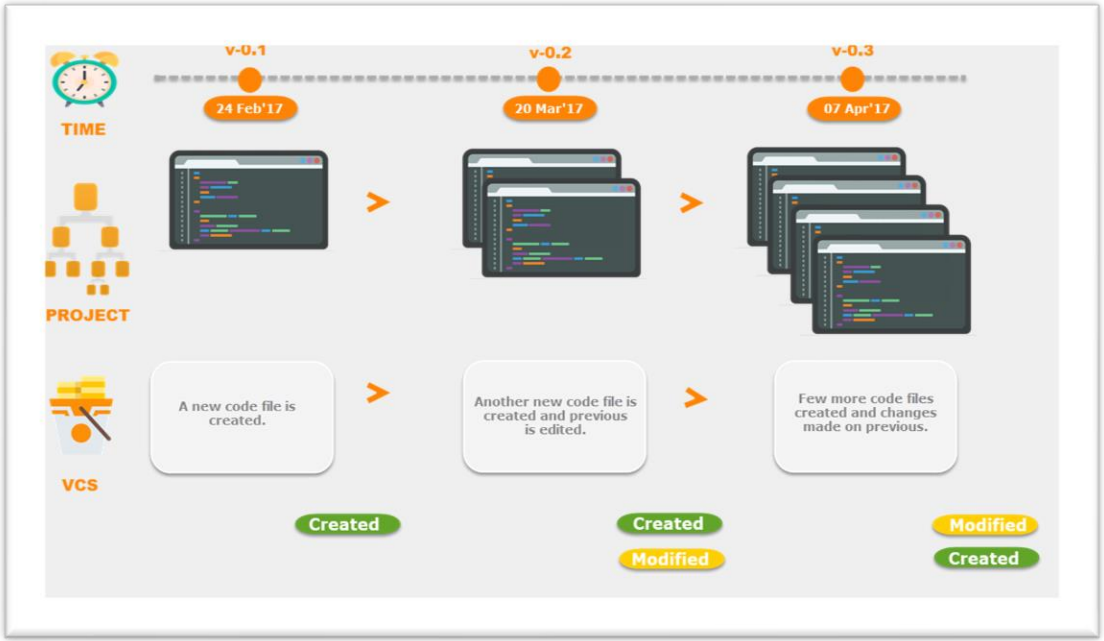
Version Control



- What is “version control”?
- Why should you care?



Contents



Version Control System

Version Control System

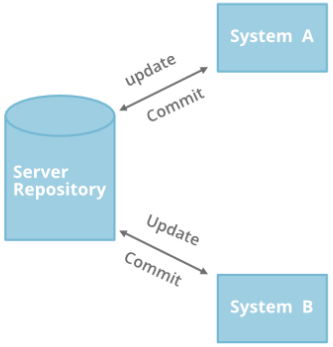
- Version control is a system that **records changes** to a file or set of files over time so that you can **recall specific versions** later

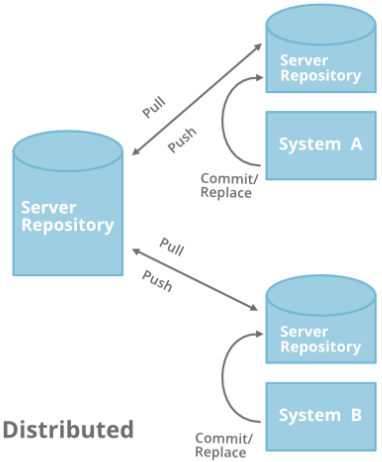
- You can control versions with **nearly any type of file**
  - software source code
  - version of an image or layout (a graphic or web designer)

Version Control System

- It allows you to:
  - **revert** selected **files** back to a previous state,
  - **revert** the entire **project** back to a previous state,
  - **compare** changes over time,
  - see **who last modified** something that might be causing a problem, who introduced an issue and when, and more.

- Using a VCS also generally means that:
  - If you screw things up or lose files, you can **easily recover**
  - In addition, you get all this for **very little overhead**





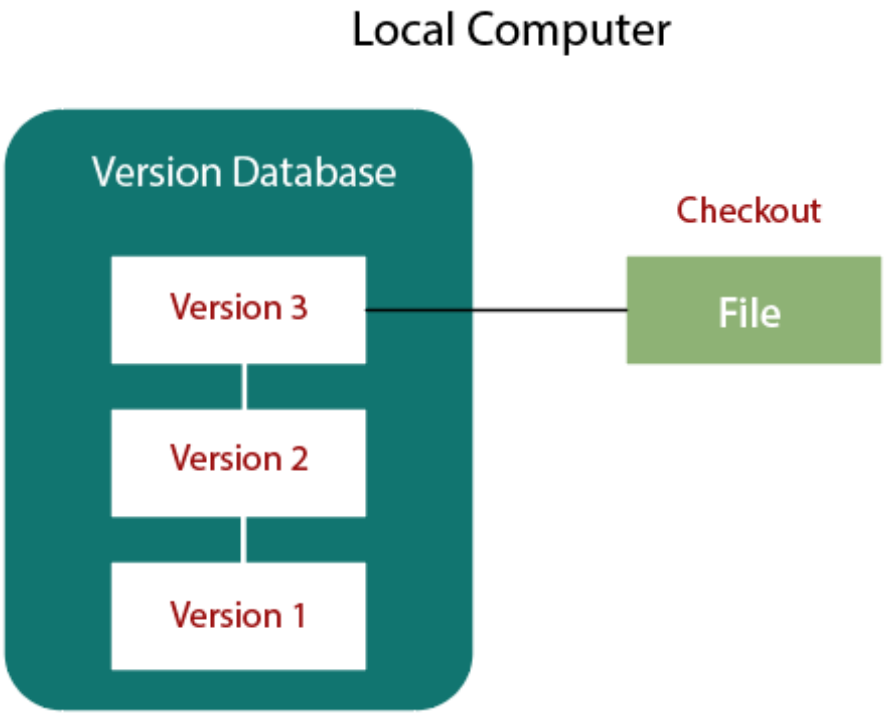
Types of version control system

Local Version Control Systems

- Many people's version control method of choice is to **copy files into another directory**
- This approach is **very common** because it is **so simple**, but it is also incredibly error prone.

It is **easy to forget** which directory you're in and **accidentally write to the wrong file** or copy over files you don't mean to

Local Version Control Systems



Local Version Control Systems

- Programmers long ago developed local VCSs that had a **simple database** that **kept all the changes** to files under revision control

- One of the most popular VCS tools was a system called RCS (Revision Control System)

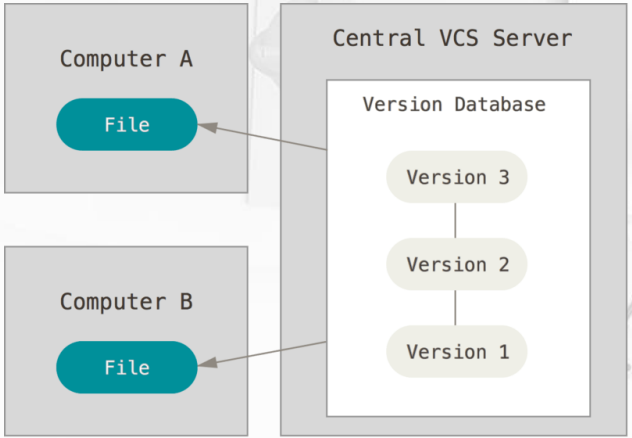
- RCS works by **keeping patch sets** in a special format on disk; it can then re-create what any file looked like at any point in time by adding up all the patches

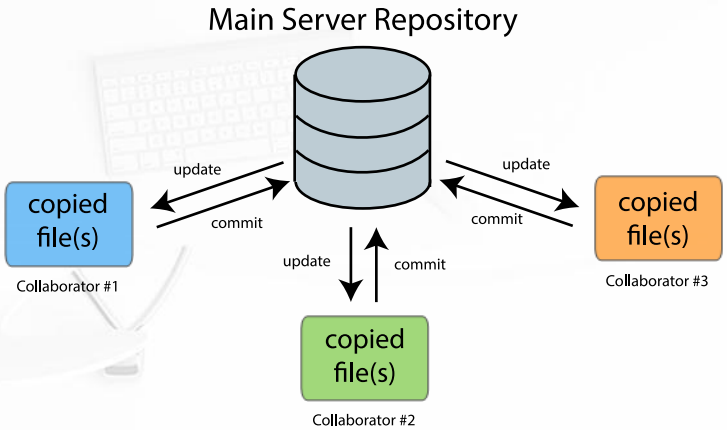
Centralized Version Control Systems

CVCS

- People need to **collaborate with developers** on other systems → CVCSs

  E.g.: CVS, Subversion, Perforce,…
- The systems **have a single server** containing all the versioned files, and a number of clients checking out files from that central place





Centralized Version Control Systems

- **Advantages:**
  - Everyone knows to a certain degree what everyone else on the project is doing
  - Administrators have fine-grained control over who can do what

- **Disadvantages:**

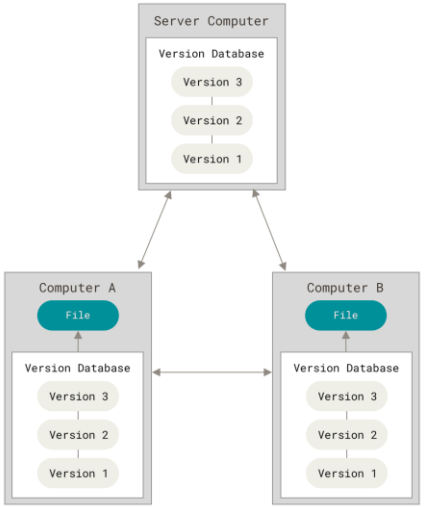
  The most obvious is the single point of failure that the **centralized server represents**
  - Servers go down
  - The server disks is corrupted

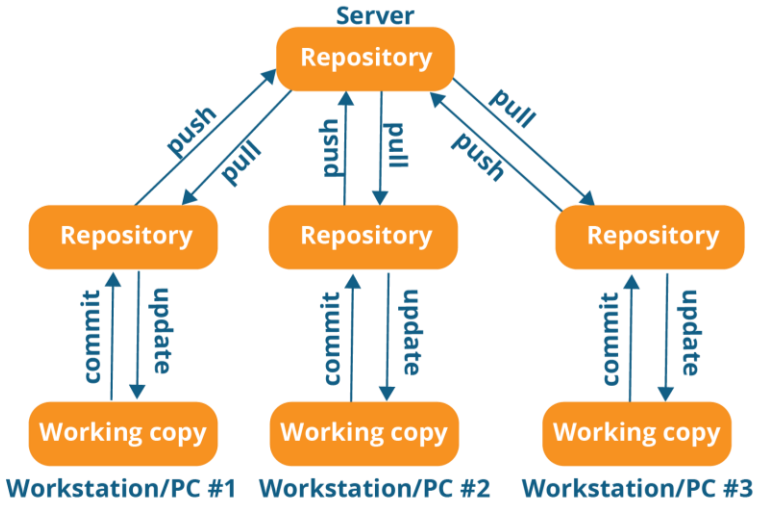
Distributed Version Control Systems

Git, Mercurial, Bazaar or Darcs

Clients check out the **latest snapshot** of the files and **fully mirror the repository** (full history)

→ If any server dies, any of the client repositories can be copied back up to the server to restore it





→ Every clone is really a full backup of all the data


git

A Short History of Git

- The Linux kernel is an open-source software project of fairly large scope

  For most of the lifetime of the Linux kernel maintenance (1991–2002), changes to the software were passed around as patches and archived files

- In 2002, the Linux kernel project began using a proprietary DVCS called BitKeeper

- In 2005, the relationship between the community that developed the Linux kernel and the commercial company that developed BitKeeper broke down

A Short History of Git

- This prompted the Linux development community to develop their own tool
- Some of the goals of the new system:
  - Speed
  - Simple design
  - Strong support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects like the Linux kernel efficiently (speed and data size)
- Since its birth in 2005, Git has evolved and matured to be easy to use and yet retain these initial qualities

What is Git?

- Git is a **free and open source** DVCS designed to handle everything from **small to very large projects** with speed and efficiency

- Git is **easy to learn** and has a tiny footprint with **lightning-fast performance**

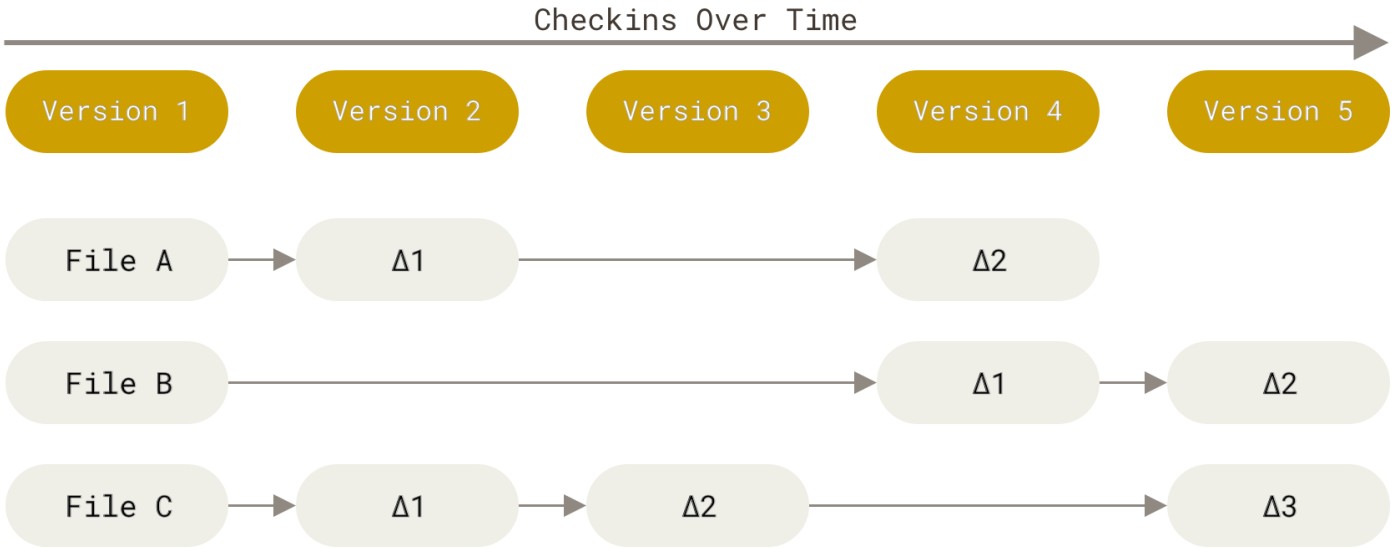
It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows

Companies & Projects Using Git

The fundamentals of how GIT works

Snapshots, Not Differences

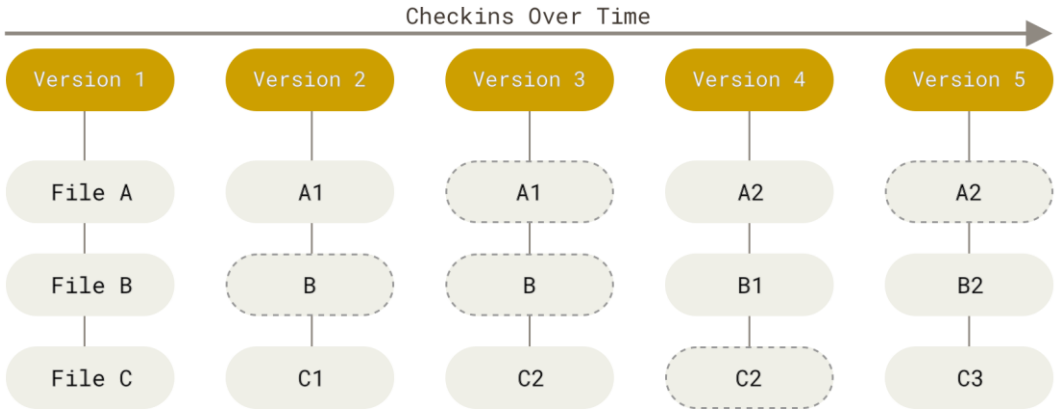
- The major difference between Git and any other VCS is the way Git thinks about its data
- Most other systems store information as a list of file-based changes



CVS, Subversion, Perforce, Bazaar

Snapshots, Not Differences

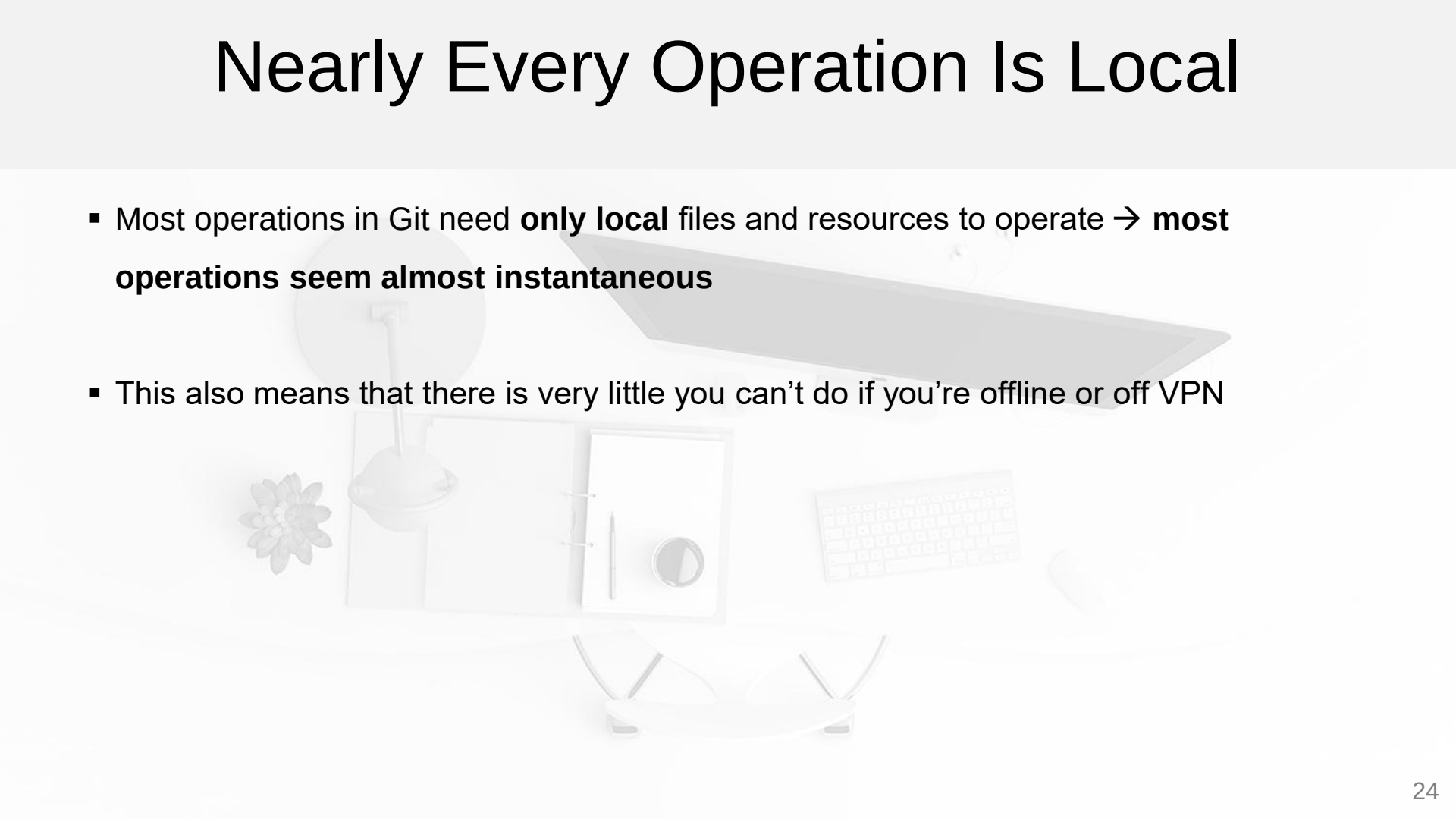
- Git thinks about its data more like a stream of snapshots
  - Every time commit, or save the state of project, Git basically takes a picture of what all files look like at that moment and stores a reference to that snapshot
  - To be efficient, if files have not changed, Git doesn't store the file again, just a link to the previous identical file it has already stored



Nearly Every Operation Is Local

- Most operations in Git need **only local** files and resources to operate → **most operations seem almost instantaneous**

- This also means that there is very little you can't do if you're offline or off VPN



Git Has Integrity

- Everything in Git is **checksummed** before it is stored and is then referred to by that checksum
- It’s impossible to change the contents of any file or directory without Git knowing about it
- The mechanism used for the checksumming is called a **SHA-1 hash**
- Git stores everything in its database not by file name but by the hash value of its contents

Git Generally Only Adds Data

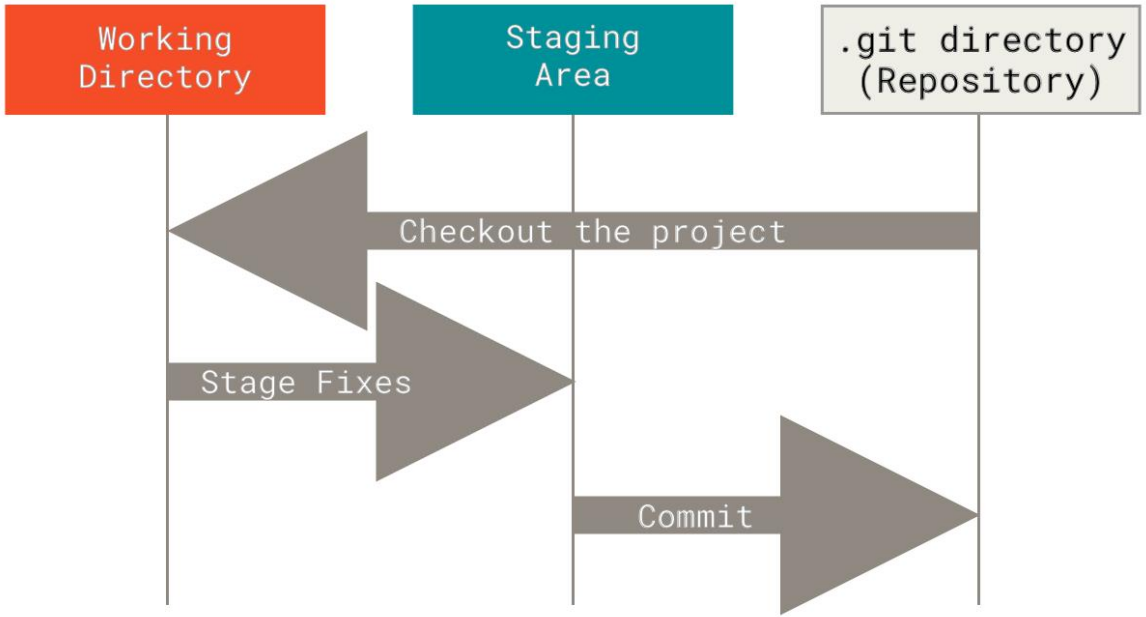
- When you do actions in Git, nearly all of them only **add** data to the Git database
- It is hard to get the system to do anything that is not undoable or to make it erase data in any way

The Three States

- Git has 3 main states that files can reside in: modified, staged, and committed
  - **Modified:** you have changed the file but have not committed it to the database yet
  - **Staged:** you have marked a modified file in its current version to go into next commit snapshot
  - **Committed:** the data is safely stored in your local database.

The Three States

- **Three main sections** of a Git project: the working tree, the staging area, and the Git directory



The Three States

- The **working tree** is a single checkout of one version. These files are pulled out of the compressed database in the Git directory and placed on disk to use or modify

- The **staging area** is a file storing information about what will go into your next commit

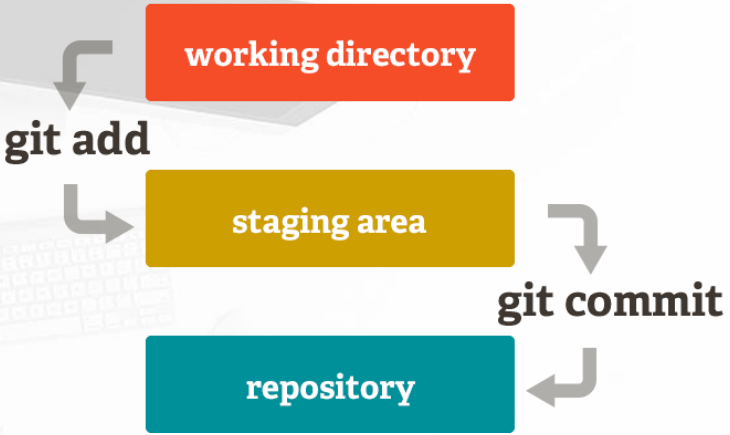
- The **Git directory** is where Git stores the metadata and object database for your project. This is what is copied when you clone a repository

Basic Git workflow

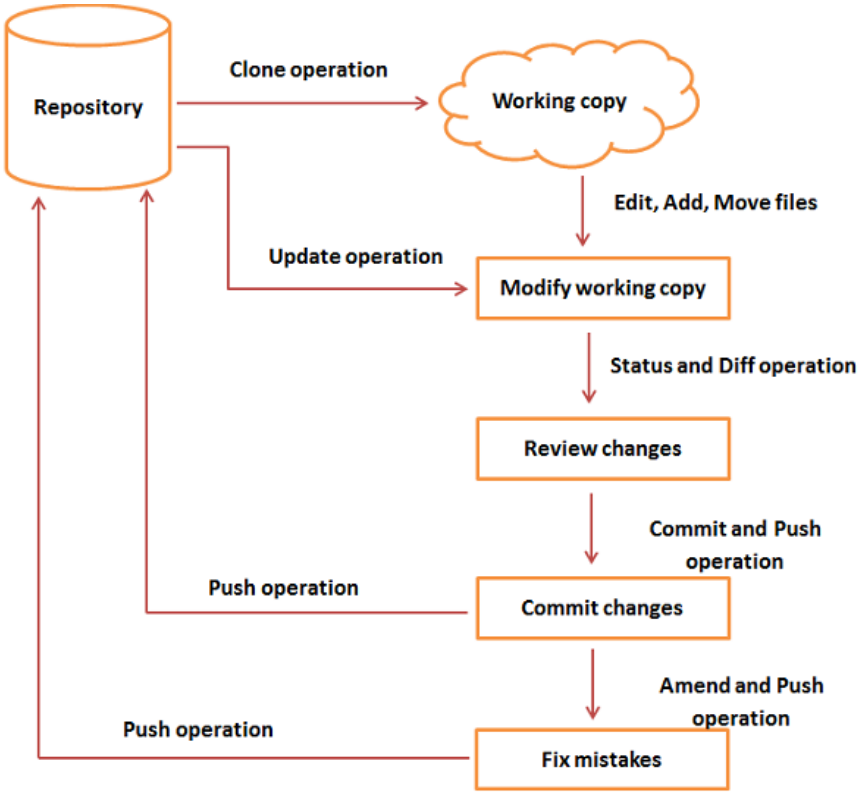
1. You **modify** files in your working tree

2. You selectively stage just those changes you want to be part of the next commit, which **adds** only those changes to the staging area

3. You do a **commit**, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory



Git Life Cycle



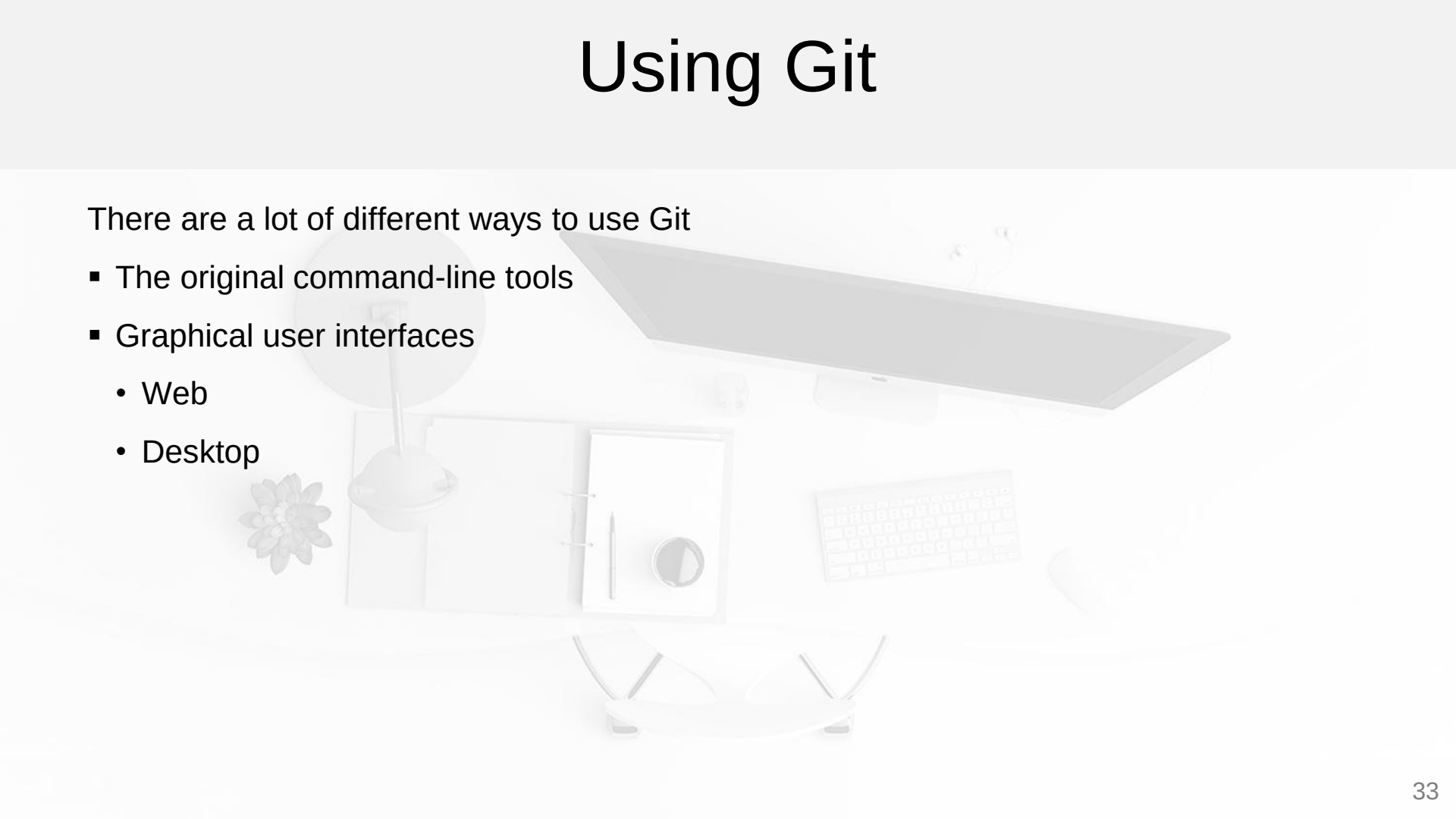
Git Life Cycle

- You **clone** the Git repository as a working copy.
- You **modify** the working copy by adding/editing files
- If necessary, you also **update** the working copy by taking other developer’s changes
- You **review** the changes before commit
- You **commit** changes. If everything is fine, then you **push** the changes to the repository.
- After committing, if you realize something is wrong, then you correct the last commit and push the changes to the repository.

Using Git

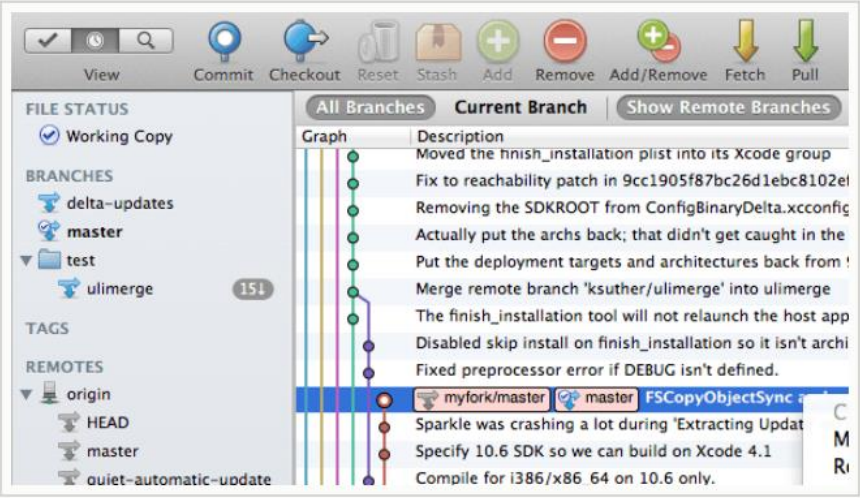
There are a lot of different ways to use Git

- The original command-line tools
- Graphical user interfaces
  - Web
  - Desktop



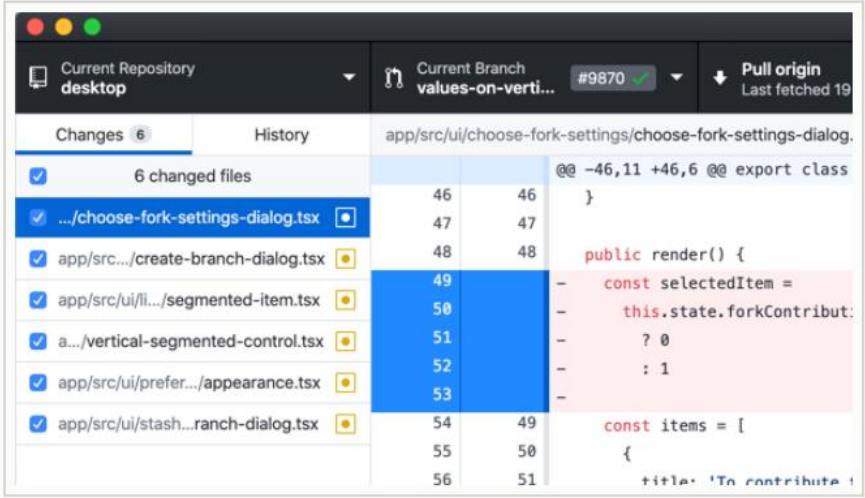
Using Git

Windows GUIs



SourceTree

Platforms: Mac, Windows
Price: Free
License: Proprietary



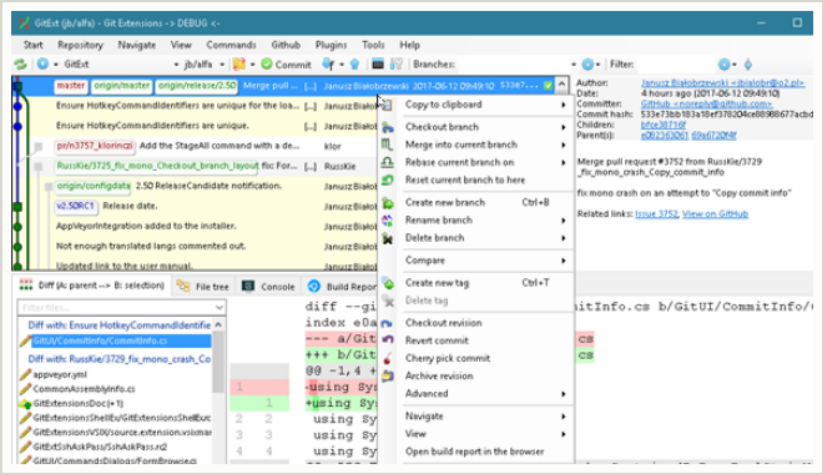
GitHub Desktop

Platforms: Mac, Windows
Price: Free
License: MIT

https://git-scm.com/download/gui/windows

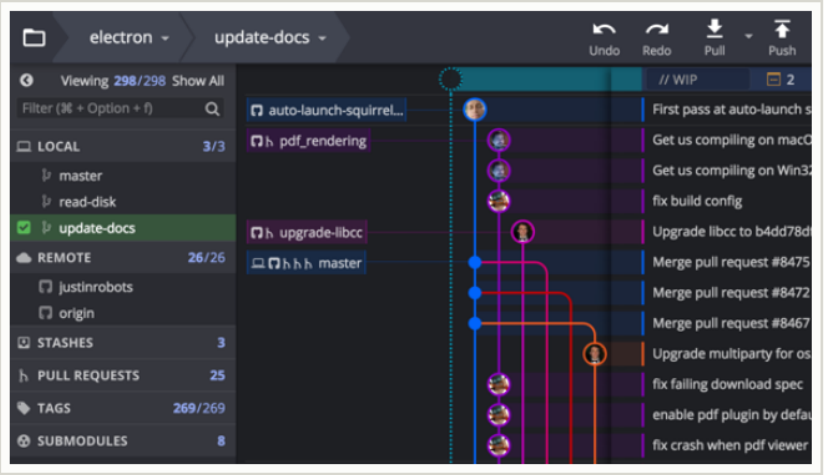
Using Git

Linux GUIs



Git Extensions

Platforms: Linux, Mac, Windows
Price: Free
License: GNU GPL



GitKraken

Platforms: Linux, Mac, Windows
Price: Free / $29 / $49
License: Proprietary

https://git-scm.com/download/gui/linux

Term Glossary

- **Repositories**

  A repository contains all of the project files (including documentation), and stores each file’s revision history

- **git**

  an open source, distributed version-control system

- **GitHub**

  a platform for hosting and collaborating on Git repositories

- **commit**

  a Git object, a snapshot of your entire repository compressed into a SHA

- **branch**

  a lightweight movable pointer to a commit

Term Glossary

- **Clone**

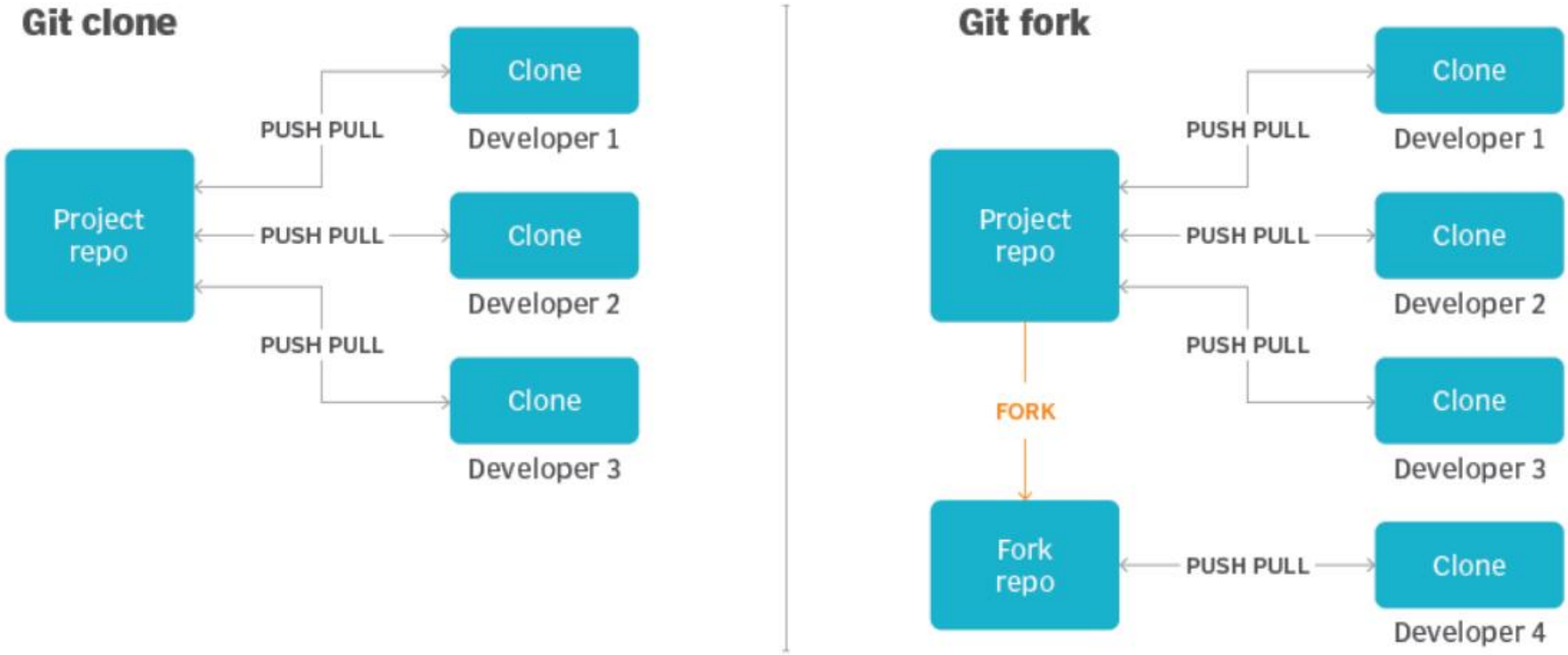
  a local version of a repository, including all commits and branches
- **Remote**

  a common repository on GitHub that all team members use to exchange their changes
- **Fork**

  a copy of a repository on GitHub owned by a different user
- **pull request**

  a place to compare and discuss the differences introduced on a branch with reviews, comments, integrated tests, and more

Clone vs Fork



Installing Git

- **Installing on Linux**

  sudo apt install git-all

- **Installing on Windows**
  - Just go to https://git-scm.com/download/win and the download will start automatically
  - Another easy way to get Git installed is by installing GitHub Desktop

Basic Git commands

- **git <verb> --help** or **man git-<verb>** get the comprehensive manual page help for the Git commands
- **git init** turns an existing directory into a new Git repository inside the folder you are running this command
- **git clone [url]** clones (downloads) a repository that already exists on GitHub, including all the files, branches, and commits
- **git add [file]** snapshots the file in preparation for versioning

Basic Git commands

- **git commit -m “[descriptive message]”** records file snapshots permanently in version history
- **git status** shows the status of changes as untracked, modified, or staged
- **git branch** shows the branches being worked on locally

Basic Git commands

- **git merge** merges lines of development together. This command is typically used to combine changes made on two distinct branches

- **git pull** updates your current local working branch with all new commits from the corresponding remote branch on GitHub

- **git push** uploads all local branch commits to the remote repository (GitHub)

Git Commands

- **Git Cheat Sheet:** https://training.github.com/downloads/github-git-cheat-sheet/
- **Complete list of all commands:** https://git-scm.com/docs/git#_git_commands
- **Git Handbook:** https://guides.github.com/introduction/git-handbook/

mercurial

About Mercurial

- Mercurial is a free, distributed source control management tool
- It efficiently handles projects of any size and offers an easy and intuitive interface

Work easier
Work faster

About Mercurial

- Mercurial is written in Python with platform independence
- Mercurial is available on Microsoft Windows, GNU/Linux, Mac OS X, Solaris 11 Express and others
- Windows users are likely to enjoy the **TortoiseHg** GUI the most

Benefit from Mercurial

- It is **fast** and **powerful**
  - Mercurial efficiently handles projects of any size and kind
  - Every clone contains the whole project history, so most actions are local, fast and convenient
  - Mercurial supports a multitude of workflows and you can easily enhance its functionality with extensions
- It is **easy to learn**
  - Simple guide to learn how to revision documents
  - Use the quick start to get going instantly
  - A short overview of Mercurial’s decentralized model is also available

Installing Mercurial

- **Installing on Linux** (Debian/Ubuntu)

  apt-get install mercurial

- **Installing on Windows**

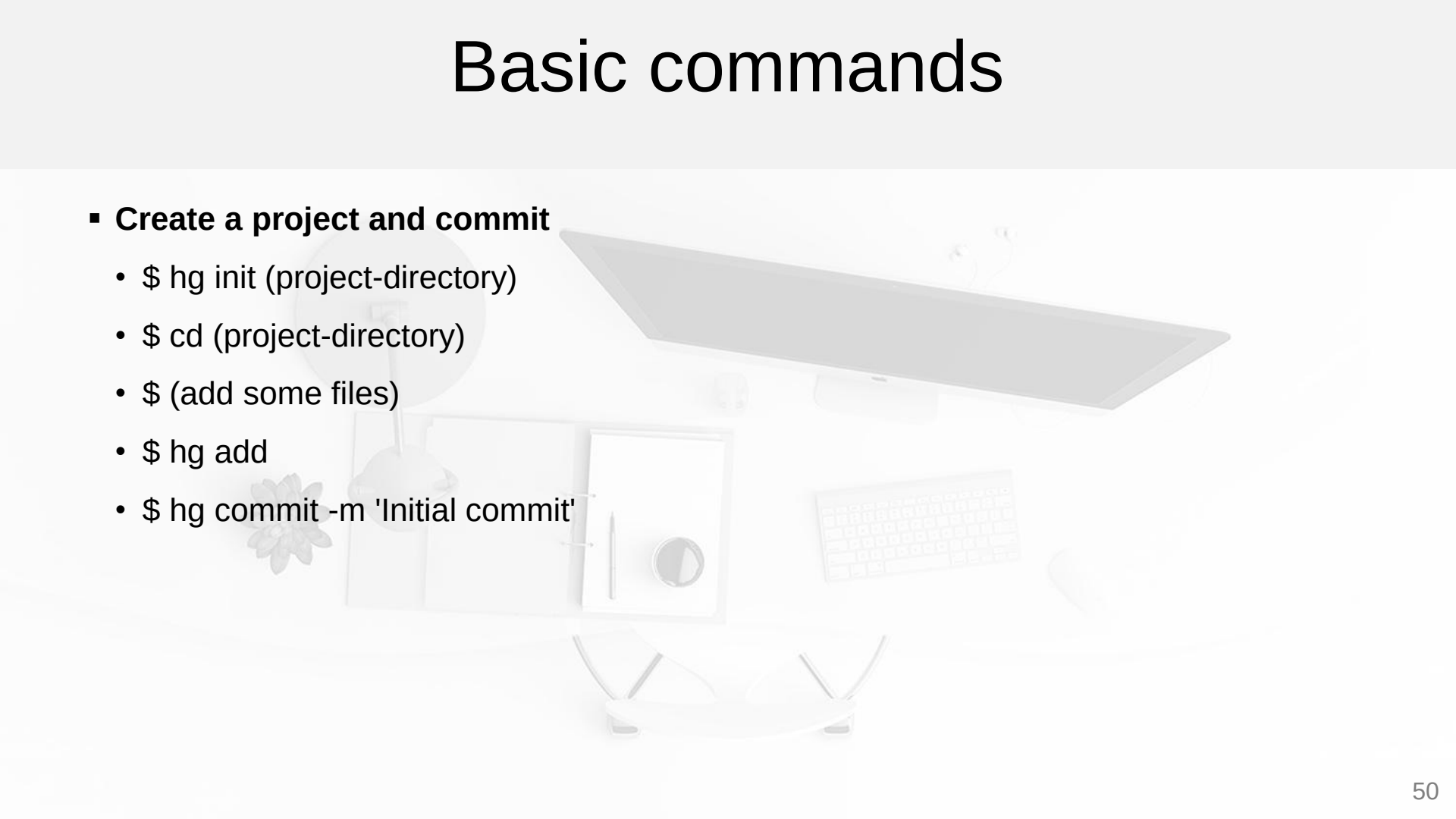
  Just go to https://www.mercurial-scm.org/downloads and download a binary package for the system

Basic commands

- **Clone a project and push changes**
  - $ hg clone https://www.mercurial-scm.org/repo/hello
  - $ cd hello
  - $ (edit files)
  - $ hg add (new files)
  - $ hg commit -m 'My changes'
  - $ hg push

Basic commands

- **Create a project and commit**
  - $ hg init (project-directory)
  - $ cd (project-directory)
  - $ (add some files)
  - $ hg add
  - $ hg commit -m 'Initial commit'



Mercurial Beginner’s Guides

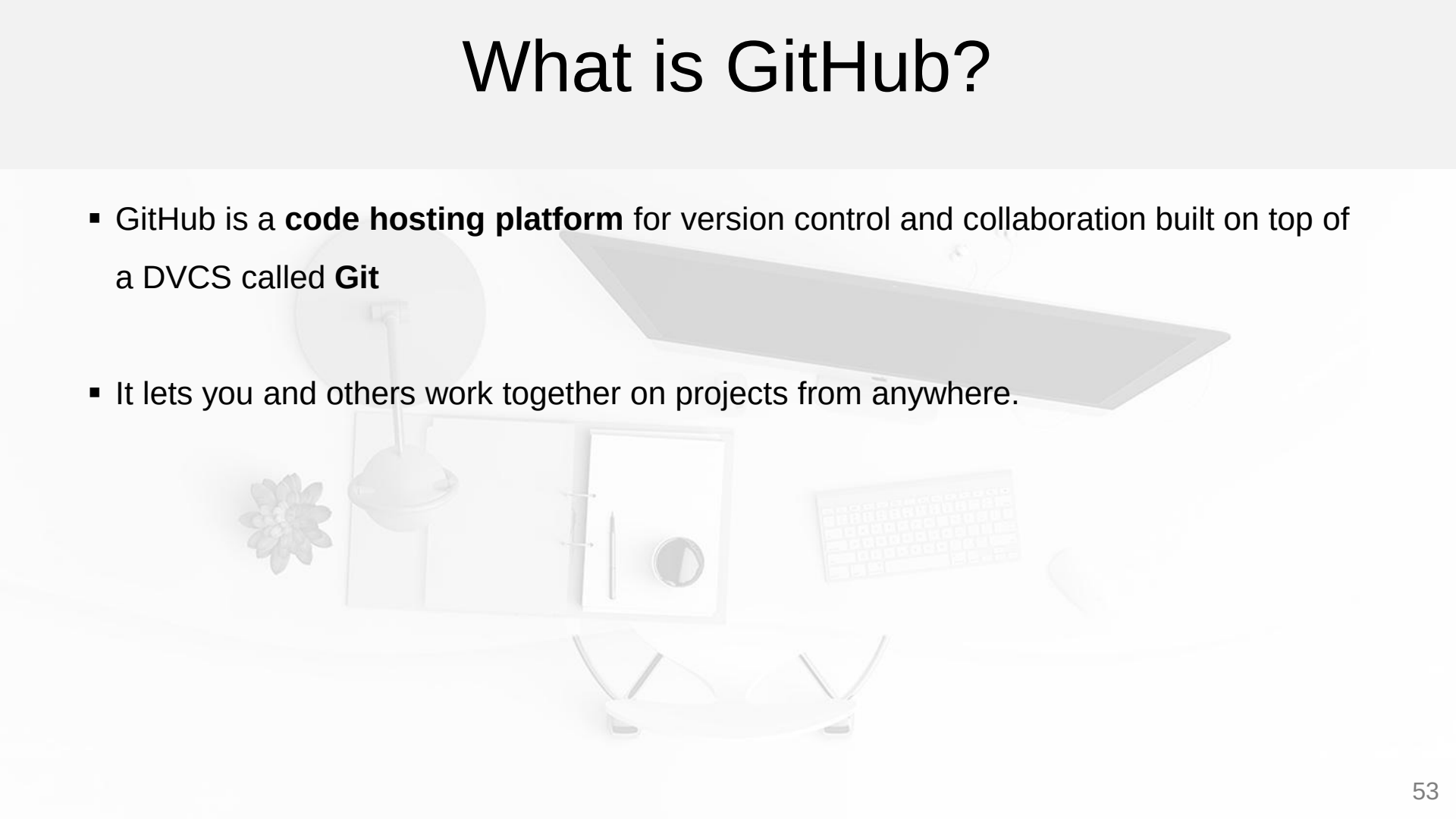
- **Understanding Mercurial**
  - A graphical illustration of basic Mercurial concepts
  - Link: https://www.mercurial-scm.org/wiki/UnderstandingMercurial
- **Tutorial**
  - A step-by-step guide for the basics
  - Link: https://www.mercurial-scm.org/wiki/Tutorial
- **QuickStart**
  - A cheat sheet for the impatient
  - Link: https://www.mercurial-scm.org/wiki/QuickStart

GitHub

What is GitHub?

- GitHub is a **code hosting platform** for version control and collaboration built on top of a DVCS called **Git**

- It lets you and others work together on projects from anywhere.



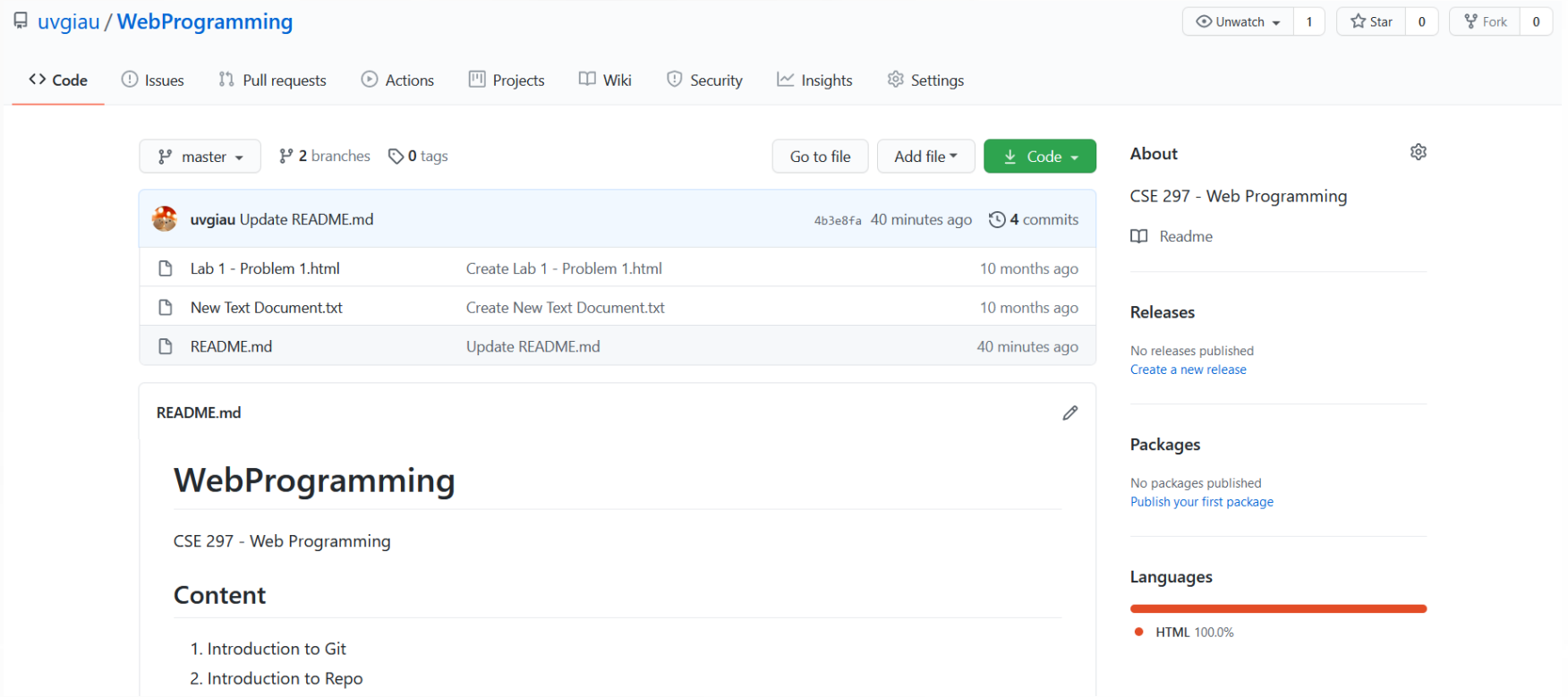
What is GitHub?

- GitHub **concentrates** on **three** things:
  - **Building a technology platform** on which developers can create, share and grow the best code possible
  - **Nurturing a community** for developers; a safe and collaborative place that facilitates sharing, amplifies creativity, and supports the principles of open source
  - **Providing access**, **opening up a community** of opportunity, where new developers can be born and where experienced developers can hone their skills and expand their knowledge

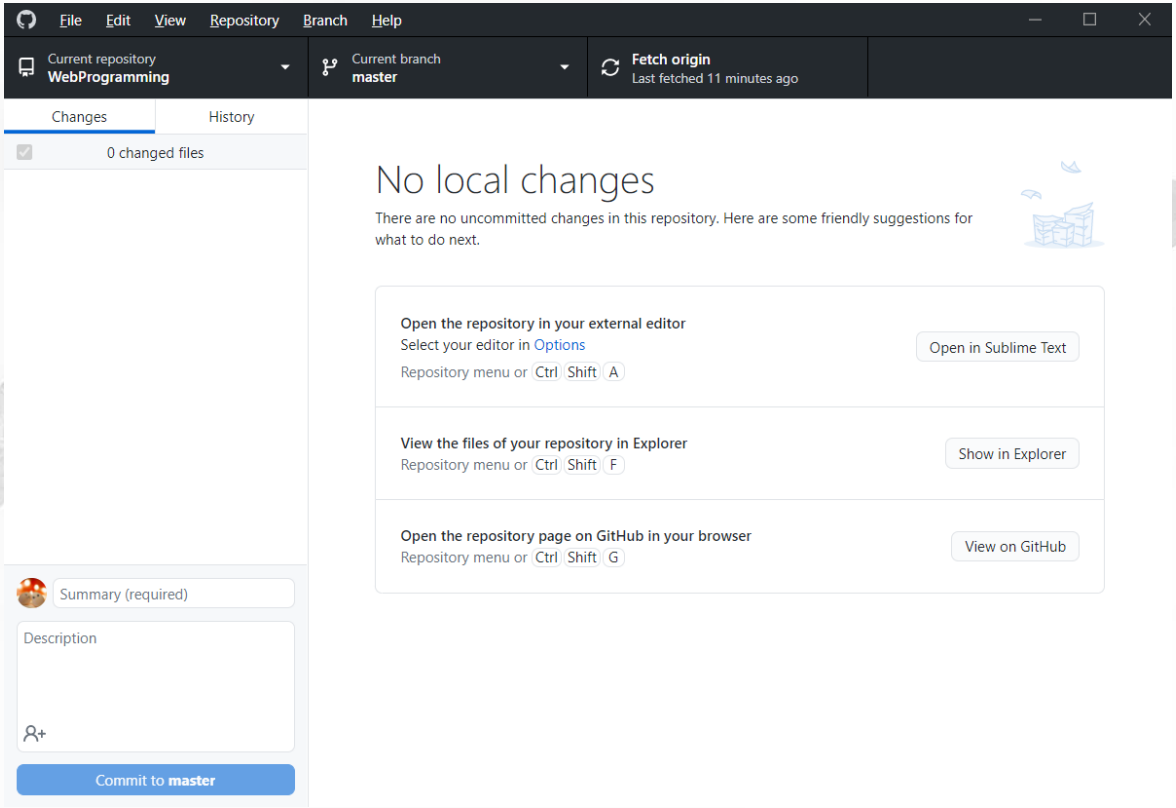
What is GitHub?

- A place to **host and share Git projects**, GitHub provides a number of features:
  - Issues: track todos, bugs, feature requests,…
  - Pull Requests: collaborate on code with other people
  - Projects: Organize issues
  - Organizations and Teams

GitHub on Web



GitHub Desktop



Markdown

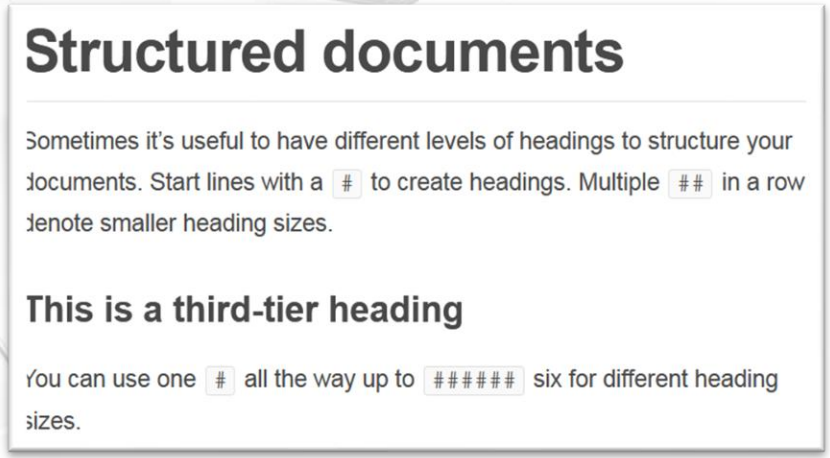
Example

```
# Structured documents

Sometimes it's useful to have different levels of headings to
structure your documents. Start lines with a `#` to create
headings. Multiple `##` in a row denote smaller heading sizes.

### This is a third-tier heading

You can use one `#` all the way up to `######` six for
different heading sizes.
```



Structured documents

Sometimes it's useful to have different levels of headings to structure your documents. Start lines with a `#` to create headings. Multiple `##` in a row denote smaller heading sizes.

This is a third-tier heading

You can use one `#` all the way up to `######` six for different heading sizes.

Markdown

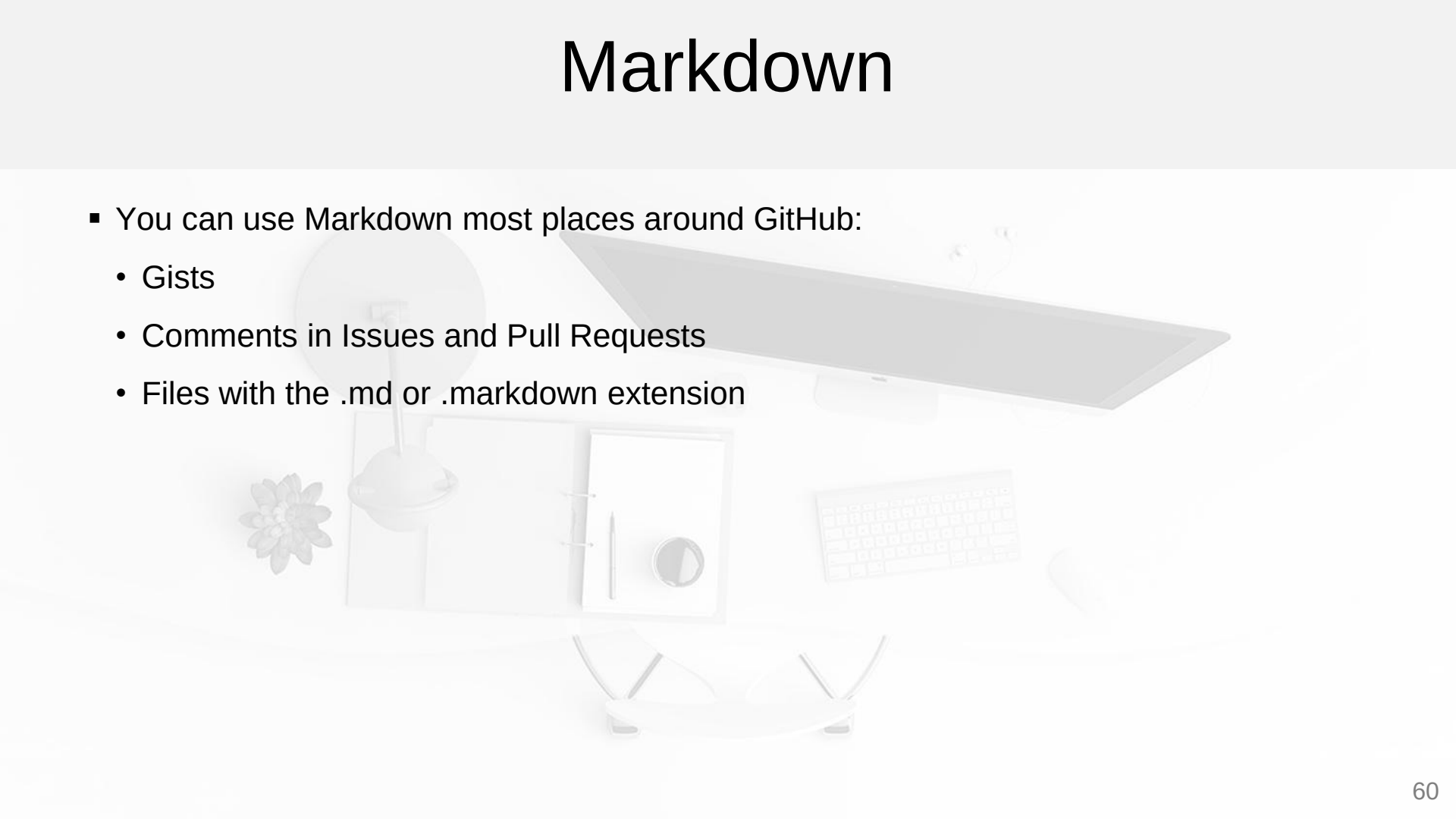
- Markdown is a way to **style text** on the web.

- You control the display of the document
  - formatting words as bold or italic,
  - adding images,
  - and creating lists

- Mostly, Markdown is **just regular text with a few non-alphabetic characters** thrown in, like # or *

Markdown

- You can use Markdown most places around GitHub:
  - Gists
  - Comments in Issues and Pull Requests
  - Files with the .md or .markdown extension



Markdown Syntax guide

- **Headers**

  # This is an <h1> tag

  ## This is an <h2> tag

  ###### This is an <h6> tag

- **Emphasis**

  *This text will be *italic**

  This will also be *italic*

  This text will be **bold**

  __This will also be **bold**__

  You **can** combine them

Markdown Syntax guide

- **Lists**
  - **Unordered**

    * Item 1
    * Item 2
      * Item 2a
      * Item 2b

  - **Ordered**

    1. Item 1
    2. Item 2
    3. Item 3
       1. Item 3a
       2. Item 3b

Markdown Syntax guide

- **Images**

  ![GitHub Logo](/images/logo.png)

  Format: ![Alt Text](url)

- **Links**

  http://github.com - automatic!

  [GitHub](http://github.com)

- **Blockquotes**

  As Kanye West said:

  > We're living the future so

  > the present is our past.

Markdown Syntax guide

- **Syntax highlighting**

  ```javascript
  function fancyAlert(arg) {
    if(arg) {
      $.facebox({div:'#foo'})
    }
  }
  ```

Markdown Syntax guide

- **Task Lists**

  - [x] @mentions, #refs, [links](), **formatting**, and ~~tags~~ supported
  - [x] list syntax required (any unordered or ordered list supported)
  - [x] this is a complete item
  - [ ] this is an incomplete item

Markdown Syntax guide

- **Tables**

  Create tables by assembling a list of words and dividing them with hyphens - (for the first row), and then separating each column with a pipe |:

```
First Header | Second Header
------------ | -------------
Content from cell 1 | Content from cell 2
Content in the first column | Content in the second column
```

ATLASSIAN

Bitbucket

About Bitbucket

- Bitbucket is a web-based version control repository hosting service owned by Atlassian for source code and development projects that use either **Mercurial** or **Git** revision control systems

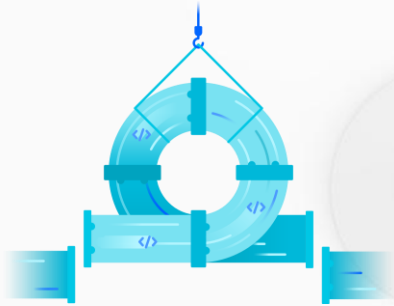
- Bitbucket offers both **commercial** plans and **free** accounts

  Free accounts: unlimited number of private repositories, up to 5 users

- Bitbucket is more than just Git code management

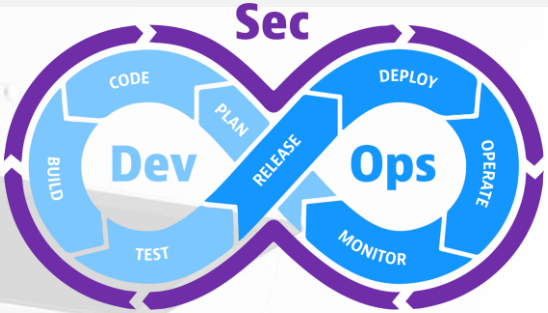
  Bitbucket gives teams one place to plan projects, collaborate on code, test, and deploy

Bitbucket Features



CI/CD built into Bitbucket

Atlassian cloud security



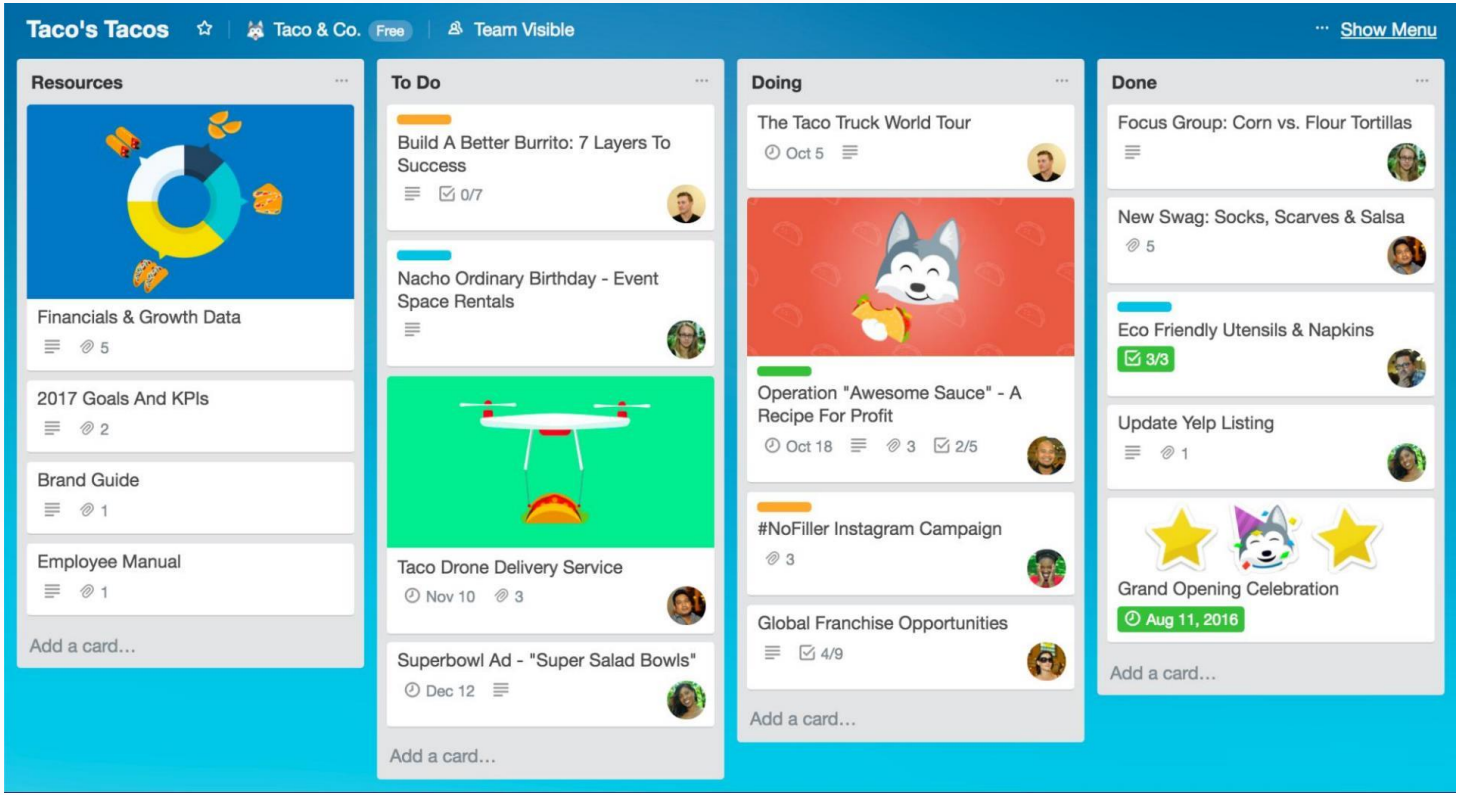
DevSecOps

Bitbucket Code Review

Jira Integration

Trello Integration

Trello

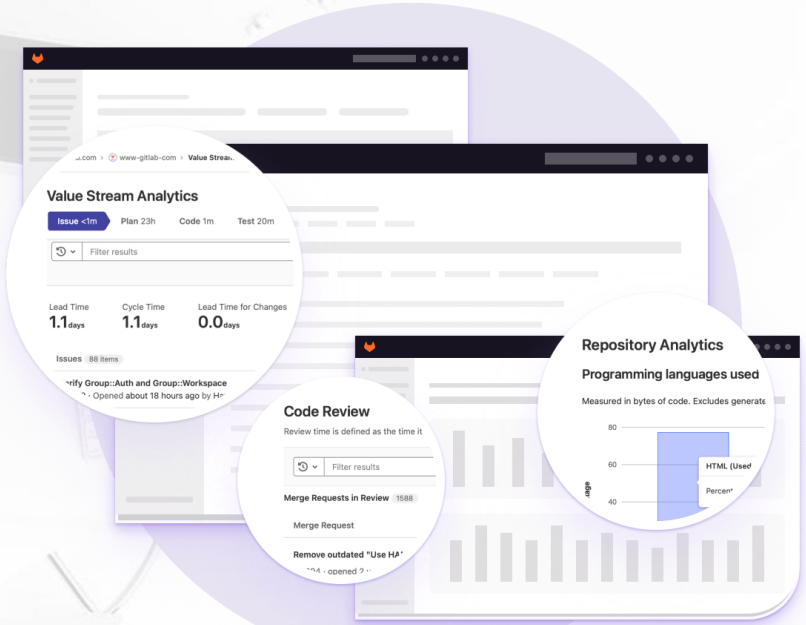

Taco's Tacos
Taco & Co.
Free
Team Visible
Show Menu
Resources
Financials & Growth Data
5
2017 Goals And KPIs
2
Brand Guide
1
Employee Manual
1
Add a card…
To Do
Build A Better Burrito: 7 Layers To Success
0/7
Nacho Ordinary Birthday - Event Space Rentals
Taco Drone Delivery Service
Nov 10
3
Superbowl Ad - "Super Salad Bowls"
Dec 12
Add a card…
Doing
The Taco Truck World Tour
Oct 5
Operation "Awesome Sauce" - A Recipe For Profit
Oct 18
3
2/5
#NoFiller Instagram Campaign
3
Global Franchise Opportunities
4/9
Add a card…
Done
Focus Group: Corn vs. Flour Tortillas
New Swag: Socks, Scarves & Salsa
5
Eco Friendly Utensils & Napkins
3/3
Update Yelp Listing
1
Grand Opening Celebration
Aug 11, 2016
Add a card…

Jira Software

GitLab

About GitLab

- GitLab is a web-based **DevOps** lifecycle tool (DevOps Platform) that provides a **Git-repository** manager providing wiki, issue-tracking and continuous integration features, using an open-source license
- The current technology stack includes Go, Ruby on Rails and Vue.js
- License: MIT



About GitLab

The One DevOps Platform for software innovation

Plan

Team Planning

Portfolio Management

Service Desk

Requirements Management

Quality Management

Design Management

Create

Source Code Management

Code Review

Wiki

Web IDE

Snippets

Verify

Continuous Integration (CI)

Code Testing and Coverage

Performance Testing

Merge Trains

Review Apps

Secrets Management

Package

Package Registry

Container Registry

Helm Chart Registry

Dependency Proxy

Git LFS

Secure

SAST

Secret Detection

Code Quality

DAST

API Security

Fuzz Testing

Dependency Scanning

Container Scanning

License Compliance

Release

Continuous Delivery

Pages

Advanced Deployments

Feature Flags

Release Evidence

Release Orchestration

Environment Management

Configure

Auto DevOps

Kubernetes Management

Deployment Management

ChatOps

Infrastructure as Code

Monitor

Metrics

Incident Management

On-call Schedule Management

Tracing

Error Tracking

Product Analytics

Govern

Security Policies

Vulnerability Management

Dependency Management

Audit Events

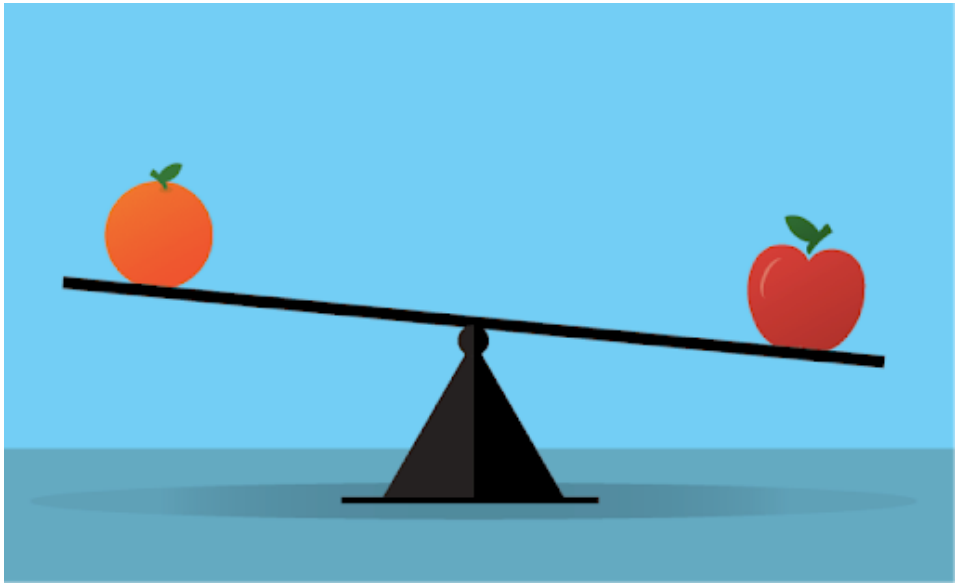
Compliance Management

Manage

Subgroups

DevOps Reports

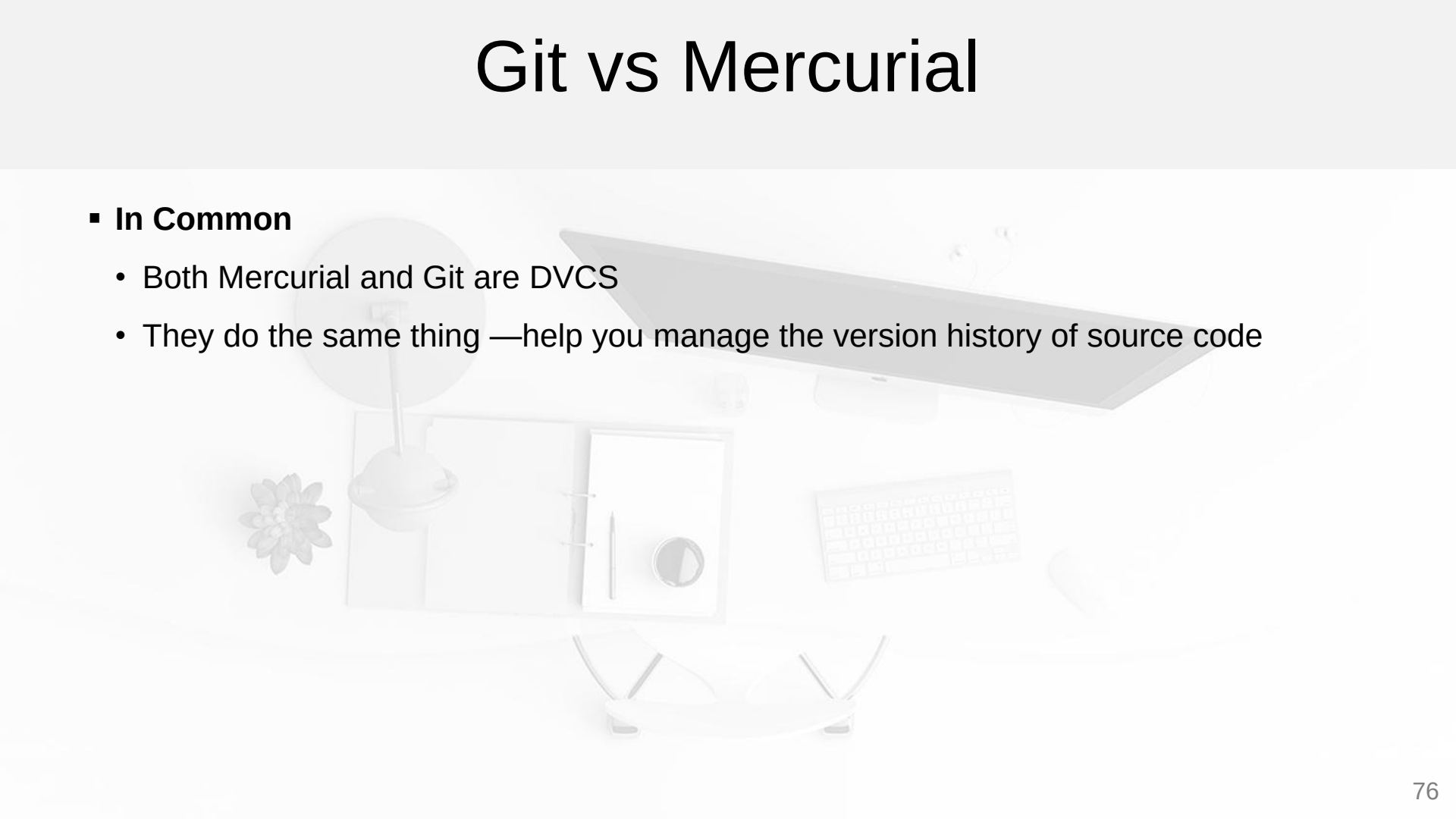
Value Stream Management



Comparison

Git vs Mercurial

- **In Common**
  - Both Mercurial and Git are DVCS
  - They do the same thing —help you manage the version history of source code



Git vs Mercurial

- **The Difference**
  - **Usability**
    - ✓Git is More Complex, with More Commands
    - ✓Mercurial is Simpler
  - **Security**
    - ✓Git is Better for Experienced Users
    - ✓Mercurial is Safer for Less Experienced Users
  - **Branching**
    - ✓Git’s Branching Model is More Effective
    - ✓Mercurial’s Branching Model can cause Confusion

GitHub vs Bitbucket vs GitLab

Free Function	GitHub	Bitbucket	GitLab
Private repo Max Collaborator	Yes Unlimited	Yes 5	Yes 5/namespace
Wiki	No (public or paid only)	Yes	Yes
Board	No (public or paid only)	No (Trello)	Yes
Capacity	15 GB (2 GB)	4 (1 GB)	5 GB
DVCS	Only Git	Not just Git, support Mercurial	Only Git
GUI	GitHub Desktop	No. Use SourceTree	No. Use SourceTree
CI/CD	2.000 minutes / mon	50 minutes / mon	400 minutes / mon

Exercise

- Sign up a GitHub Account
- Install GitHub Desktop
- Try using some basic functions


Q&A

References

- **GitHub vs Bitbucket: Which is Right for Your Development Team?**

  https://www.elegantthemes.com/blog/wordpress/github-vs-bitbucket
- **GitHub Vs GitLab Vs Bitbucket**

  https://www.gangboard.com/blog/github-vs-gitlab-vs-bitbucket
- **Bitbucket vs GitHub vs GitLab**

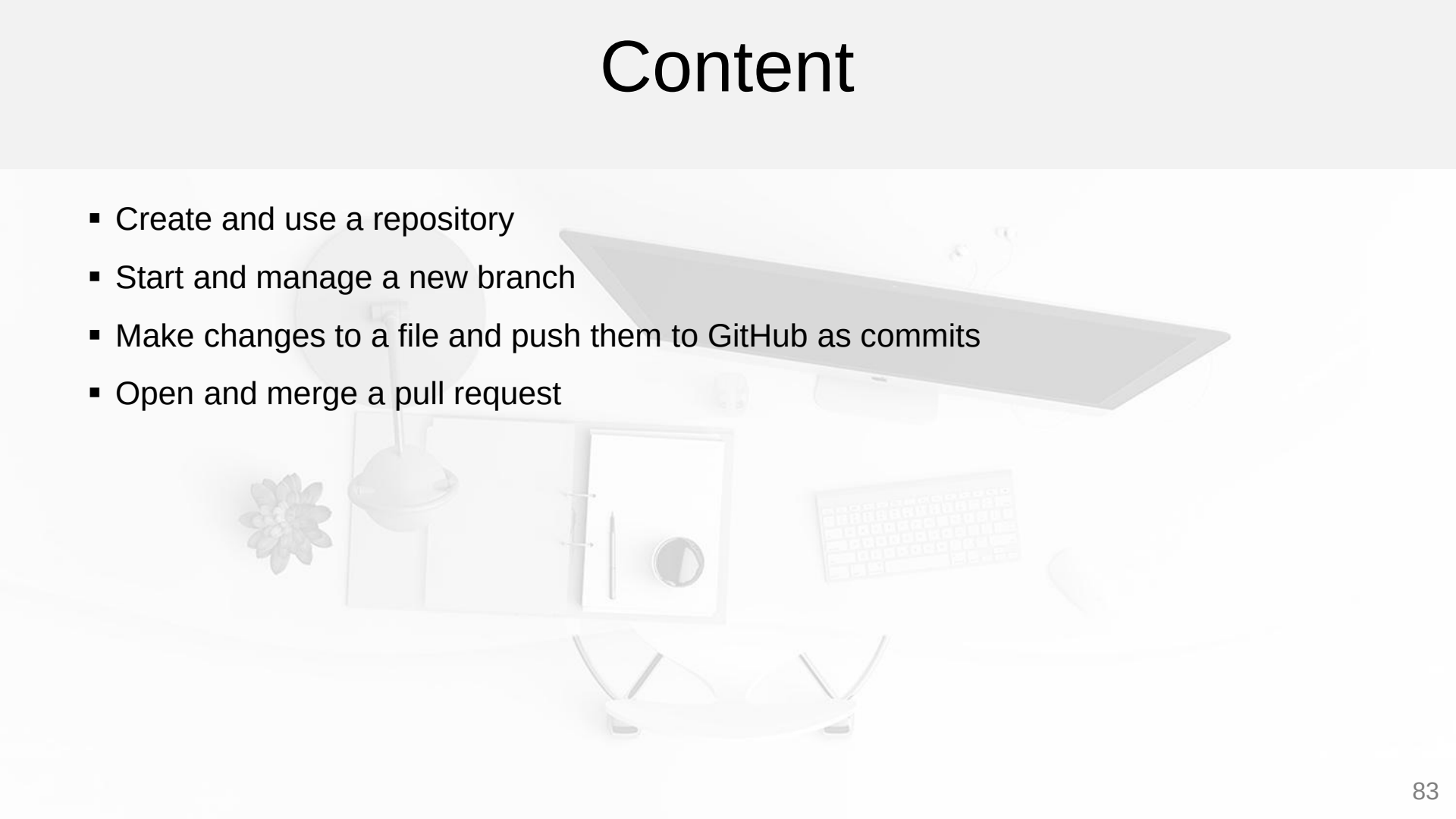
  https://www.geeksforgeeks.org/bitbucket-vs-github-vs-gitlab/

GitHub on Web

Hello World Tutorial

Content

- Create and use a repository
- Start and manage a new branch
- Make changes to a file and push them to GitHub as commits
- Open and merge a pull request



Step 1. Create a Repository

Review

- A **repository** is usually used to organize a single project.

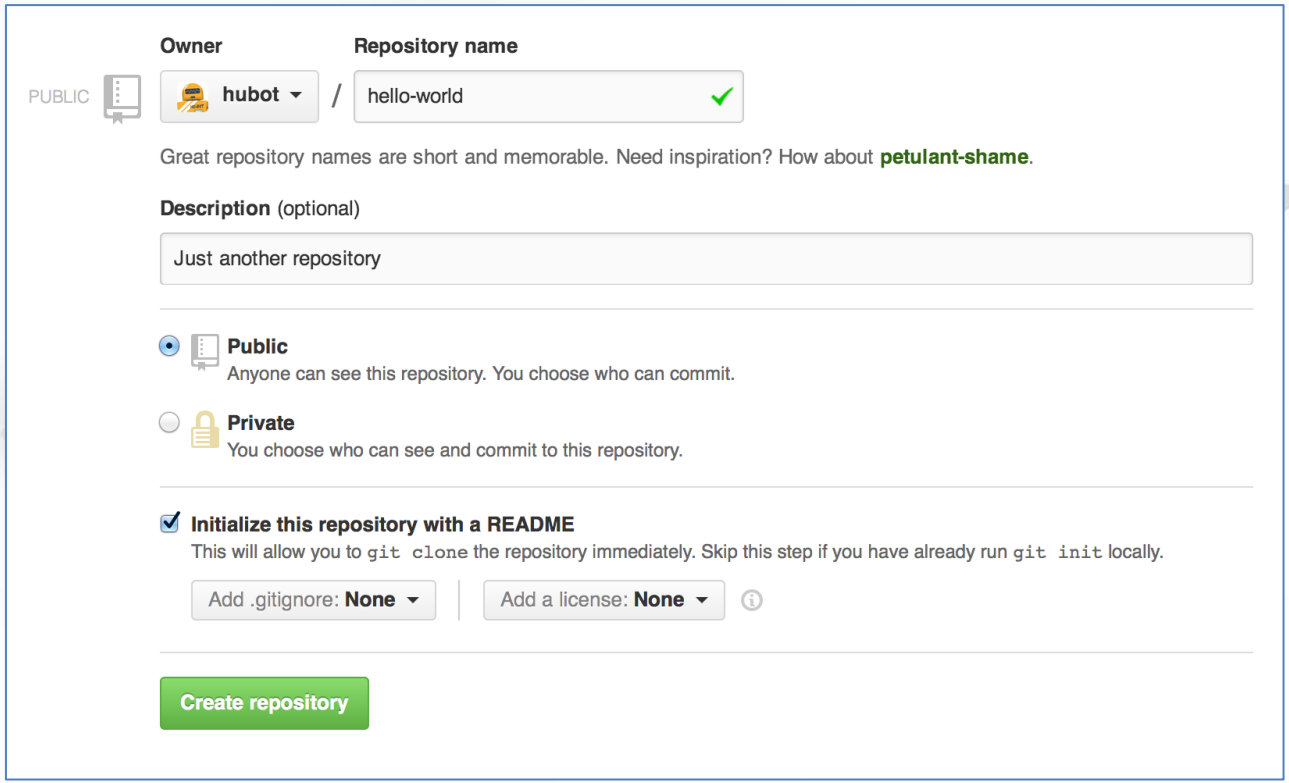
- Repositories can contain folders and files, images, videos, spreadsheets, and data sets – anything your project needs

Step 1. Create a Repository

To create a new repository

1. In the upper right corner, next to your avatar or identicon, click **+** and then select **New repository**.
2. Name your repository *hello-world*.
3. Write a short description.
4. Select Initialize this repository with a **README**
5. Click **Create repository**.

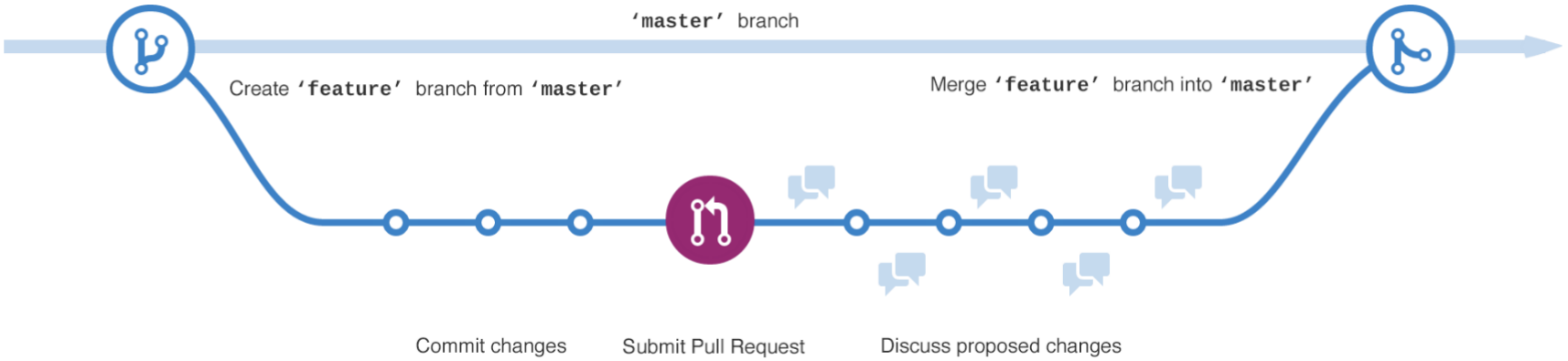
Step 1. Create a Repository



Step 2. Create a Branch

Review

- **Branching** is the way to work on different versions of a repository at one time
- The **main** branch is the definitive branch
- When you create a branch off the main branch, you're making a copy, or snapshot, of main as it was at that point in time.

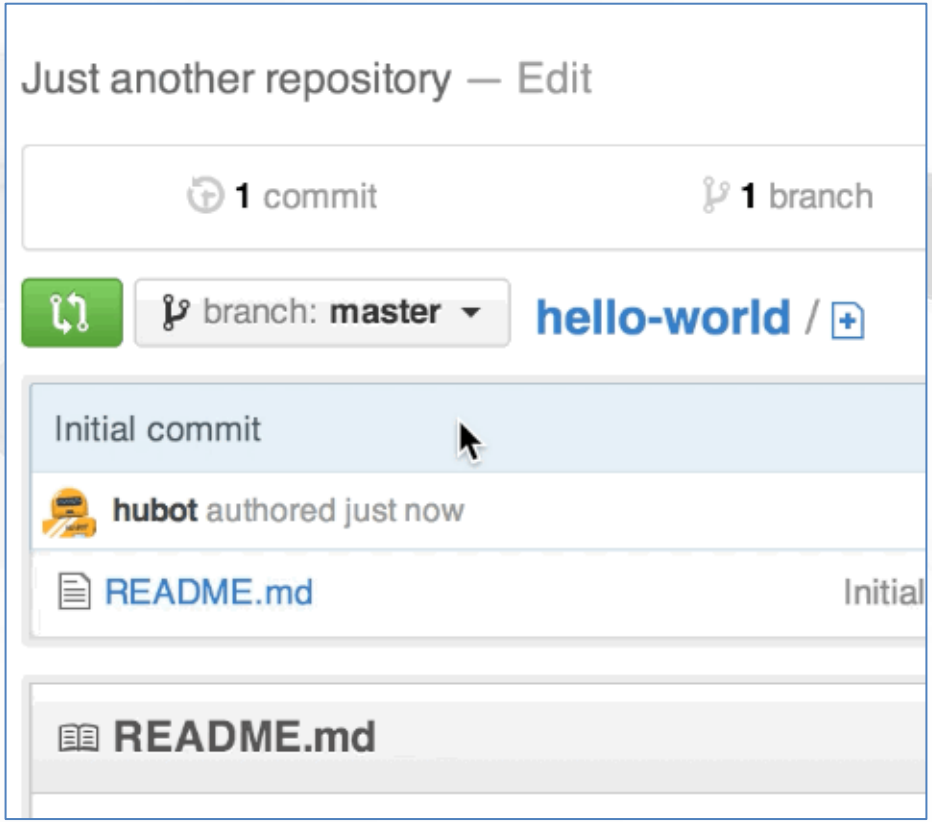


Step 2. Create a Branch

To create a new branch

1. Go to your new repository *hello-world*
2. Click the drop down at the top of the file list that says **branch: main**
3. Type a branch name, *readme-edits*, into the new branch text box
4. Select the blue **Create branch** box or hit “Enter” on your keyboard.

Step 2. Create a Branch



Step 3. Make and commit changes

Review

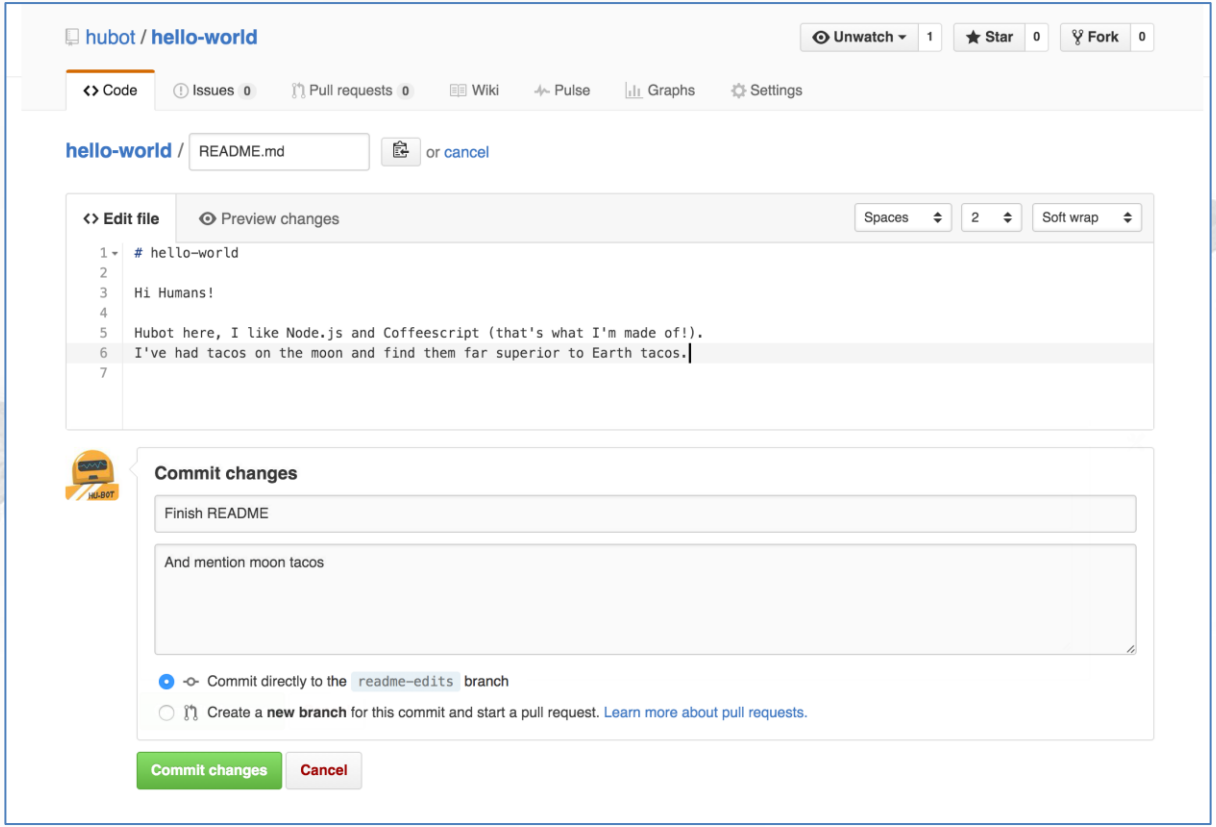
- On GitHub, saved changes are called **commits**
- Each commit has an associated **commit message**, which is a description explaining why a particular change was made
- Commit messages capture the history of your changes, so other contributors can understand **what you've done and why**

Step 3. Make and commit changes

Make and commit changes

1. Click the *README.md* file
2. Click the ✎ pencil icon in the upper right corner of the file view to edit
3. In the editor, write a bit about yourself
4. Write a commit message that describes your changes
5. Click **Commit changes** button

Step 3. Make and commit changes



Step 4. Open a Pull Request

Review

- When you open a **pull request**, you’re proposing your changes and requesting that someone review and pull in your contribution and merge them into their branch

- Pull requests show **diffs**, or differences, of the content from both branches. The changes, additions, and subtractions are shown in green and red

Step 4. Open a Pull Request

Open a Pull Request for changes to the README

1. Click the **Pull Request** tab, then from the Pull Request page, click the green **New pull request** button
2. In the **Example Comparisons** box, select the branch you made, *readme-edits*, to compare with *main* (the original)
3. Look over your changes in the diffs on the Compare page, make sure they're what you want to submit.
4. When you're satisfied that these are the changes you want to submit, click the big green **Create Pull Request** button
5. Give your pull request a title and write a brief description of your changes
6. When you're done with your message, click **Create pull request**!

Step 5. Merge your Pull Request

1. Click the green **Merge pull request** button to merge the changes into *main*
2. Click **Confirm merge**
3. Go ahead and delete the branch, since its changes have been incorporated, with the **Delete branch** button in the purple box