

GIT – DISTRIBUTED VERSION CONTROL SYSTEM

Jeff Gregory, P.E.
Civil Engineer
Remote Sensing/GIS, CRREL



US Army Corps
of Engineers

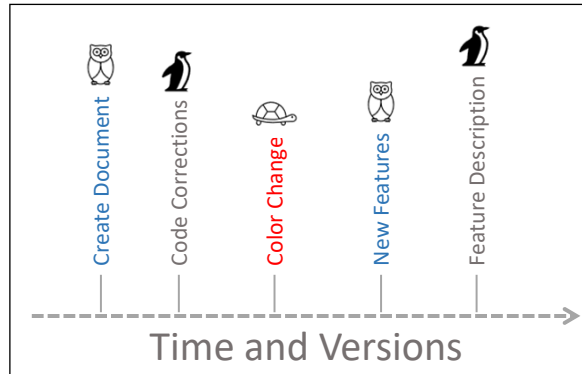


Git Basics



What is Version Control?

- A system that records changes to a file or set of files over time so that you can recall specific versions later
- History of content changes
- Facilitate collaborative changes
 - Who, when and why changes are made



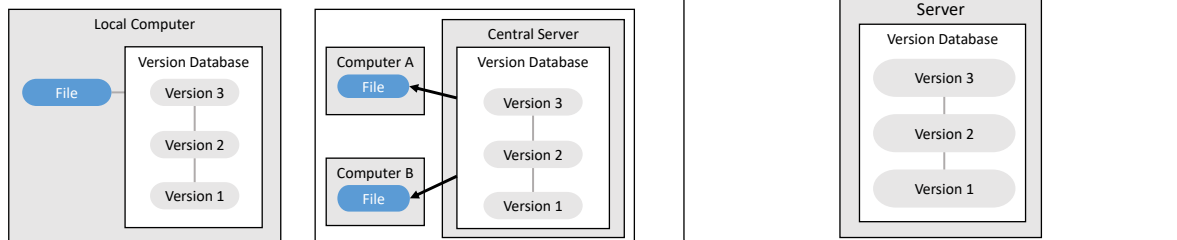
Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

Keep track of a history of changes over time

Facilitate collaboration with a history of who made the change, when changes were made, and descriptions why those changes were made

Version Control Types

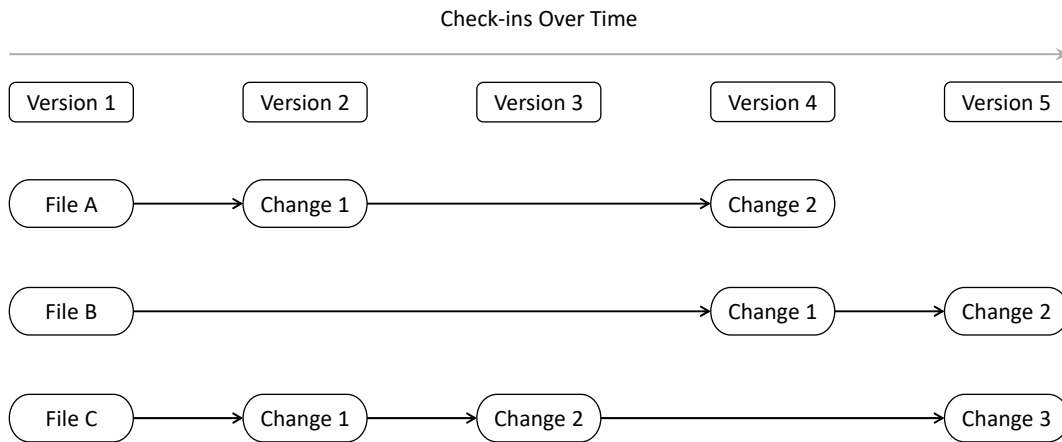
- Local Version Control
- Centralized Version Control
- Distributed Version Control



Types of version control:

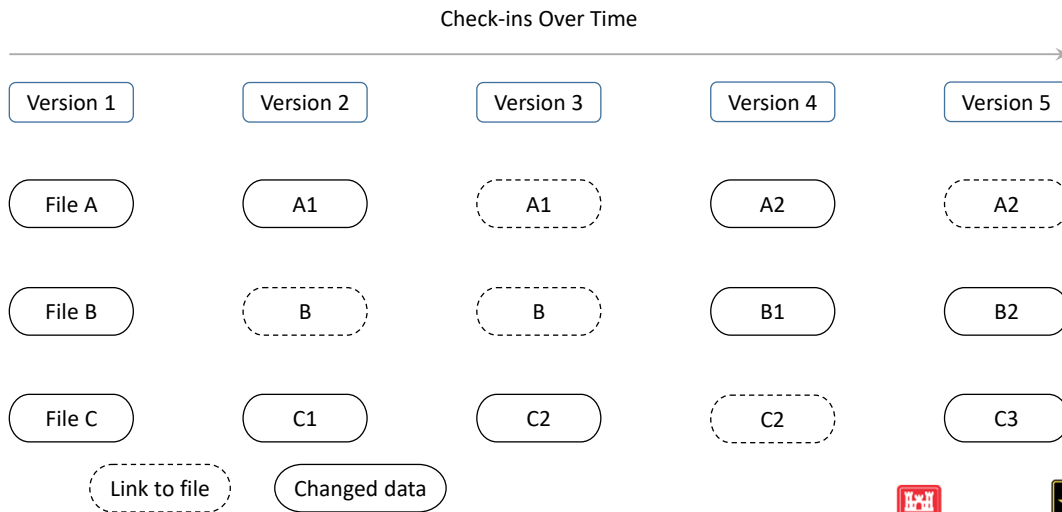
- Local Version Control can be simply someone copying files to another directory
- Centralized Version Control have a single server containing all versioned files that clients check out files from the central location; server goes down and everything could be lost
- Distributed Version Control, which is Git, doesn't just check out file changes but a fully mirrors the repository; if a server dies, any client repository can be used to restore

Snapshots, Not Differences



The major difference between Git and any other VCS (Subversion and friends included) is the way Git thinks about its data. Conceptually, most other systems store information as a list of file-based changes. These other systems (CVS, Subversion, Perforce, Bazaar, and so on) think of the information they store as a set of files and the changes made to each file over time (this is commonly described as **delta-based** version control).

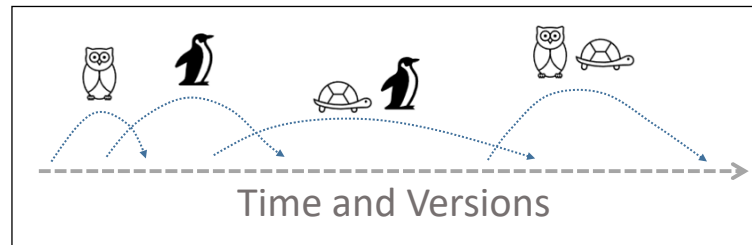
Snapshots, Not Differences



Git doesn't think of or store its data this way. Instead, Git thinks of its data more like a series of snapshots of a miniature filesystem. With Git, every time you commit, or save the state of your project, Git basically takes a picture of what all your 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. Git thinks about its data more like a **stream of snapshots**

What is Git?

- Nearly every operation is local (locally enabled version control)
- Git Integrity
- Git Generally Only Adds Data
- Distributed system that clients fully mirror the repository



Nearly Every Operation is Local:

Most operations in Git need only local files and resources to operate — generally no information is needed from another computer on your network. If you're used to a CVCS where most operations have that network latency overhead, this aspect of Git will make you think that the gods of speed have blessed Git with unworldly powers. Because you have the entire history of the project right there on your local disk, most operations seem almost instantaneous.

Git Integrity:

Everything in Git is checksummed before it is stored and is then referred to by that checksum. This means it's impossible to change the contents of any file or directory without Git knowing about it. This functionality is built into Git at the lowest levels and is integral to its philosophy. You can't lose information in transit or get file corruption without Git being able to detect it.

a 40-character string composed of hexadecimal characters (0-9 and a-f)

The mechanism that Git uses for this checksumming is called a SHA-1 hash.

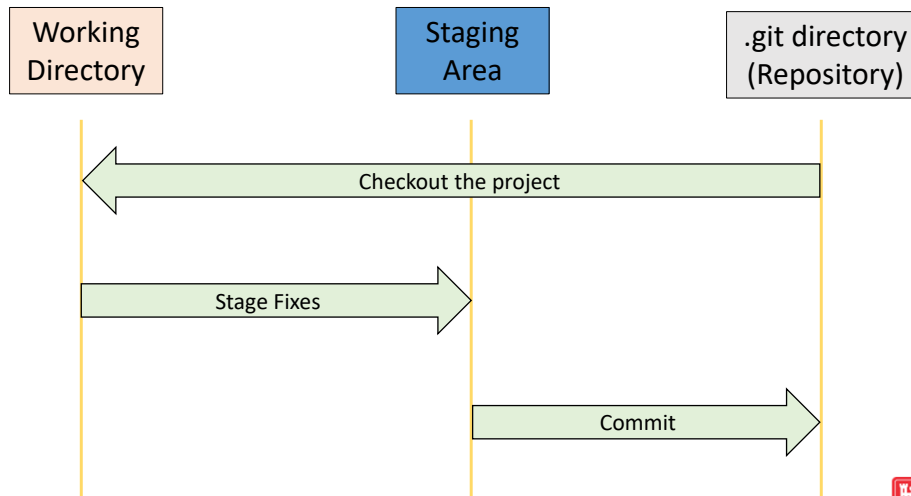
Git Generally Only Adds Data:

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. As with any VCS, you can lose or mess up changes you haven't committed yet, but after you commit a snapshot into Git, it is very difficult to lose, especially if you regularly push your database to another repository.

Distributed System:

Acting like a backup having the full mirrored repository on the server and local machines. This also allows for collaboration working freely on any file at any time.

Three Main Stages



Git has three main states that your files can reside in: **modified**, **staged**, and **committed**:

- Modified means that you have changed the file but have not committed it to your database yet.
- Staged means that you have marked a modified file in its current version to go into your next commit snapshot.
- Committed means that the data is safely stored in your local database.

This leads us to the three main sections of a Git project: the working tree, the staging area, and the Git directory.

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

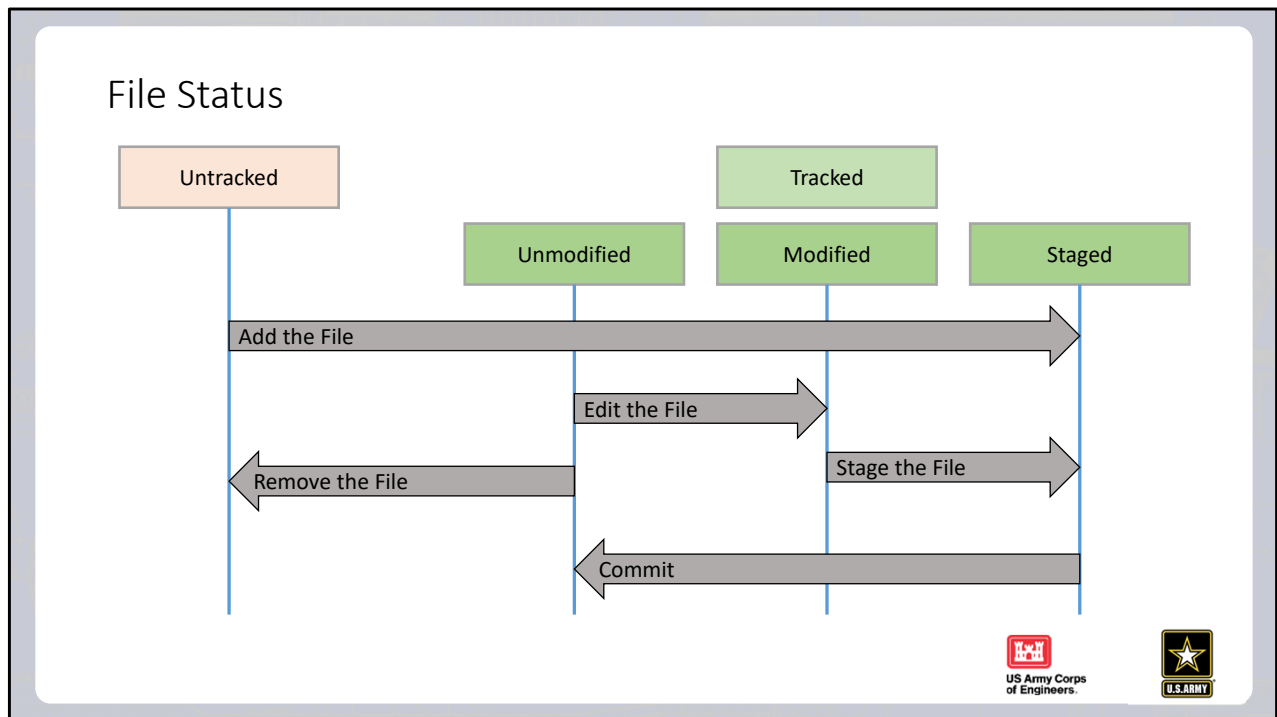
The staging area is a file, generally contained in your Git directory, that stores information about what will go into your next commit. The Git directory is where Git stores the metadata and object database for your project. This is the most important part of Git, and it is what is copied when you **clone** a

repository from another computer.

The basic Git workflow goes something like this:

1. You modify files in your working tree.
2. You selectively stage just those changes you want to be part of your 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.

If a particular version of a file is in the Git directory, it's considered **committed**. If it has been modified and was added to the staging area, it is **staged**. And if it was changed since it was checked out but has not been staged, it is **modified**.



Remember that each file in your working directory can be in one of two states: tracked or untracked. Tracked files are files that were in the last snapshot, as well as any newly staged files; they can be unmodified, modified, or staged.

So, tracked files are files that Git knows about. Untracked files are everything else
→ Any files in your working directory that were not in your last snapshot and are not in your staging area.

When you first clone a repository, all of your files will be tracked and unmodified because Git just checked them out and you haven't edited anything.

As you edit files, Git sees them as modified, because you've changed them since your last commit. As you work, you selectively stage these modified files and then commit all those staged changes, and the cycle repeats.

Git Basic Commands

\$ Getting Help

```
$ git help <verb>  
$ git <verb> -h  
$ man git-<verb>
```

\$ git config

```
$ git config --global user.name "name"  
$ git config --global user.email "name@example.com"  
$ git config --global init.defaultBranch main  
$ git config --list [--show-origin]
```

\$ git init [directory]

\$ git clone <url> [target directory]

\$ git add <pathspec>

```
$ git add .  
$ git add -A
```

\$ git status

```
$ git status -s
```

\$ git commit

```
$ git commit -m "message"  
$ git commit -a
```

\$ git log

```
$ git log --oneline --graph
```

\$ git rm

```
$ git rm --cached < pathspec >
```

\$ git mv <source> <destination>

\$ git pull [<repository> [<refspec>...]]

\$ git push [<repository> [<refspec>...]]

\$ git branch

\$ git checkout

\$ git remote



Git Installation and Setup

Installation on Windows, VS Code and first-time setup



Git Installation on Windows

- <https://www.git-scm.com/download/win>
 - 32-bit or 64-bit Git for Windows **Portable**
 - Save as... | remove “exe” | Save as type “All files (*.*)”
 - Extract 7z file content (C:\apps\Git\)
- Setup Git for CMD
 - Go to “Edit environment variables for your account”
 - Add C:\git\path\to\bin to “Path”
 - Open command prompt testing at the prompt
 - git version
 - where git



First-Time Git Setup

- Command tool `git config`
- `~/.gitconfig` or `~/.config/git/config` file
- On Windows systems, Git looks for the `.gitconfig` file in the `$HOME` directory
- Config file in the Git directory (`.git/config`)
- Your Identity not credentials

```
$ git config --list [--show-origin]
```

```
$ git config --global user.name "John Doe"
```

```
$ git config --global user.email johndoe@example.com
```

- Your editor
- Checking keys

```
$ git config --global core.editor "C:\path\to\editor"
```

```
git config [--show-origin] <key>
```

```
$ git config user.name
```



Command tool `git config` lets you get and set configuration variables that control all aspects of how Git looks and operates

Configuration file `~/.gitconfig` or `~/.config/git/config`: Values specific personally to you, the user. You can make Git read and write to this file specifically by passing the `--global` option, and this affects **a11** of the repositories you work with on your system

Config file in the Git directory: Specific to that single repository. You can force Git to read from and write to this file with the `--local` option, which is the default option.

Identity: The first thing you should do when you install Git is to set your user name and email address. This is important because every Git commit uses this information, and it's immutably baked into the commits you start creating

Editor:

Checking keys:

Git Setup in VS Code

- Start VS Code
- Open Settings UI
 - File | Preferences | Settings or
 - Ctrl+,
- In “Search Settings” type “git path”
- Under “Git: path” click “Edit in settings.json”
- Add “git.path”:
“C:\\path\\to\\git.exe”
- Integrate Bash Terminal
 - Open Settings UI
 - In “Search Settings” type “terminal integrated profiles”
 - Under “Terminal > Integrated > Profiles: Windows” click “Edit in settings.json”
 - Add the following

```
"terminal.integrated.profiles.windows": {  
  "Command Prompt": {  
    "path": ["C:\\apps\\Git\\git-cmd.exe"],  
    "args": ["--command=\\.\\bin\\bash.exe", "--login", "-i"]  
  }  
},
```



Get Started with Git and GitHub

Hello World Exercise



GitHub: Hello World

- Code hosting platform for version control and collaboration
- Create a Hello World Repository
- README file
- <https://www.markdownguide.org/cheat-sheet/>
- 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



Code hosting platform and VC: GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere

Create a Hello World Repo: Cover GitHub essentials like repositories, branches, commits, and pull requests

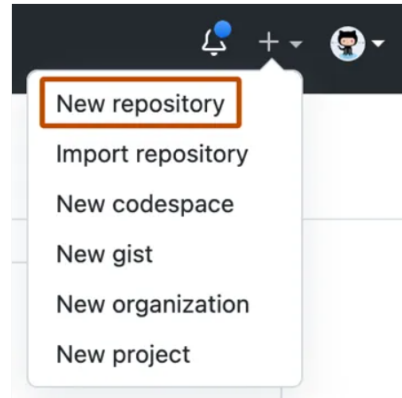
- A repository is usually used to organize a single project
- Repositories can contain folders and files, images, videos, spreadsheets, and data sets

README: Often, repositories include a README file, a file with information about your project


- README files are written in the plain text Markdown language
- <https://www.markdownguide.org/cheat-sheet/>

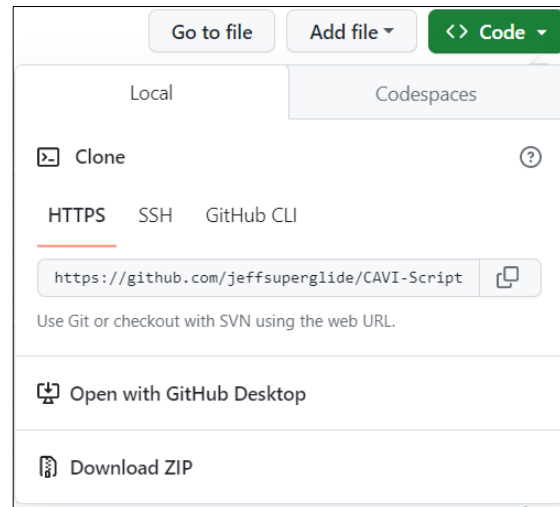
Creating a Repository

- In the upper-right corner of any page, use the drop-down menu, and select New repository
- In the "Repository name" box, type hello-world
- In the "Description" box, type a short description
- Select whether your repository will be **Public** or **Private**
- Select **Add a README file**
- Click **Create repository**



Cloning a Repository

- Select “Code”
- Select HTTPS
- Copy the url 
- On a command line
 - `git clone <url>`
- Paste the URL
- The repository will default to the “main” branch



Create a Branch

- Make sure to be in the “hello-world” directory
- On the command line, “git branch” to list available branches
- On the command line, “git checkout -b readme-edits” to create a new branch named “readme-edits” and switches to it
- Branching - different versions of a repository at one time
- By default, your repository has one branch named “main”, the definitive branch



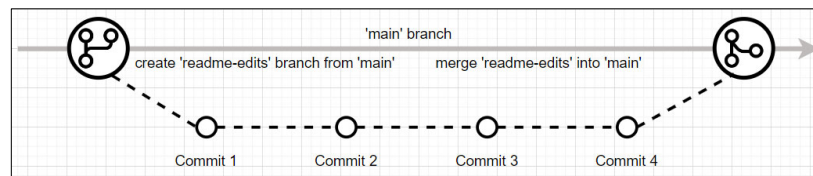
Branching: Branching lets you have different versions of a repository at one time

One Branch Name: By default, your repository has one branch named “main” that is considered to be the definitive branch

- You can create additional branches off of main in your repository
- You can use branches to have different versions of a project at one time
- Work done on different branches will not show up on the main branch until you merge it
- Use branches to experiment and make edits before committing them to main

Making and Committing Changes

- Under the readme-edits branch you created, edit the README file using VS Code
- Update the README file with plain text and/or Markdown elements
 - <https://loremipsum.io/> for text generator
- Stage changes
 - `$ git add README.md`
- Commit changes
 - `$ git commit -m "message"`
- Push changes
 - `$ git push origin readme-edits`



Opening a Pull Request in GitHub

- Click the Pull requests tab of your hello-world repository
- Click New pull request
- Comparing changes: Select readme-edits to compare with main
- Look over your changes in the diffs on the Compare page, make sure they're what you want to submit
- Click **Create pull request**
- Give your pull request a title and write a brief description of your changes. You can include emojis and drag and drop images and gifs
- Click **Create pull request**



Merging Pull Request in GitHub

- Click the Pull requests tab of your hello-world repository
- At the bottom of the pull request, click **Merge pull request** to merge the changes into main
- Click **Confirm merge**
- Click **Delete branch**



Getting a Git Repository

- Initializing a repository in an existing directory

```
$ git init
```

- Initializing a repository in a new directory

```
$ git init <directory name>
```

- Cloning an existing repository

```
$ git clone <url>
```

```
$ git clone https://github.com/jeffsuperglide/CAVI-Script-Training.git
```

- Branch Naming

```
$ git config --global init.defaultBranch <name>
```

```
$ git branch -m <name>
```



You typically obtain a Git repository in one of two ways:

1. You can take a local directory that is currently not under version control, and turn it into a Git repository, or
2. You can clone an existing Git repository from elsewhere.

In either case, you end up with a Git repository on your local machine, ready for work.

Initializing in a directory: This creates a new subdirectory named `.git` that containing all of your necessary repository files — a Git repository skeleton. At this point, nothing in your project is tracked yet.

Initializing a new directory: This create a new directory named `<directory name>` and a subdirectory named `.git`

Cloning: To get a copy of the existing Git repository. This will create a directory in you current location named `<repo name>`

Branch Naming: To configure the initial branch name to use in all of your new repositories. The just-created branch can be renamed via `git branch -m <name>`