

# Version Control with Git

# Session Agenda: Version Control with Git

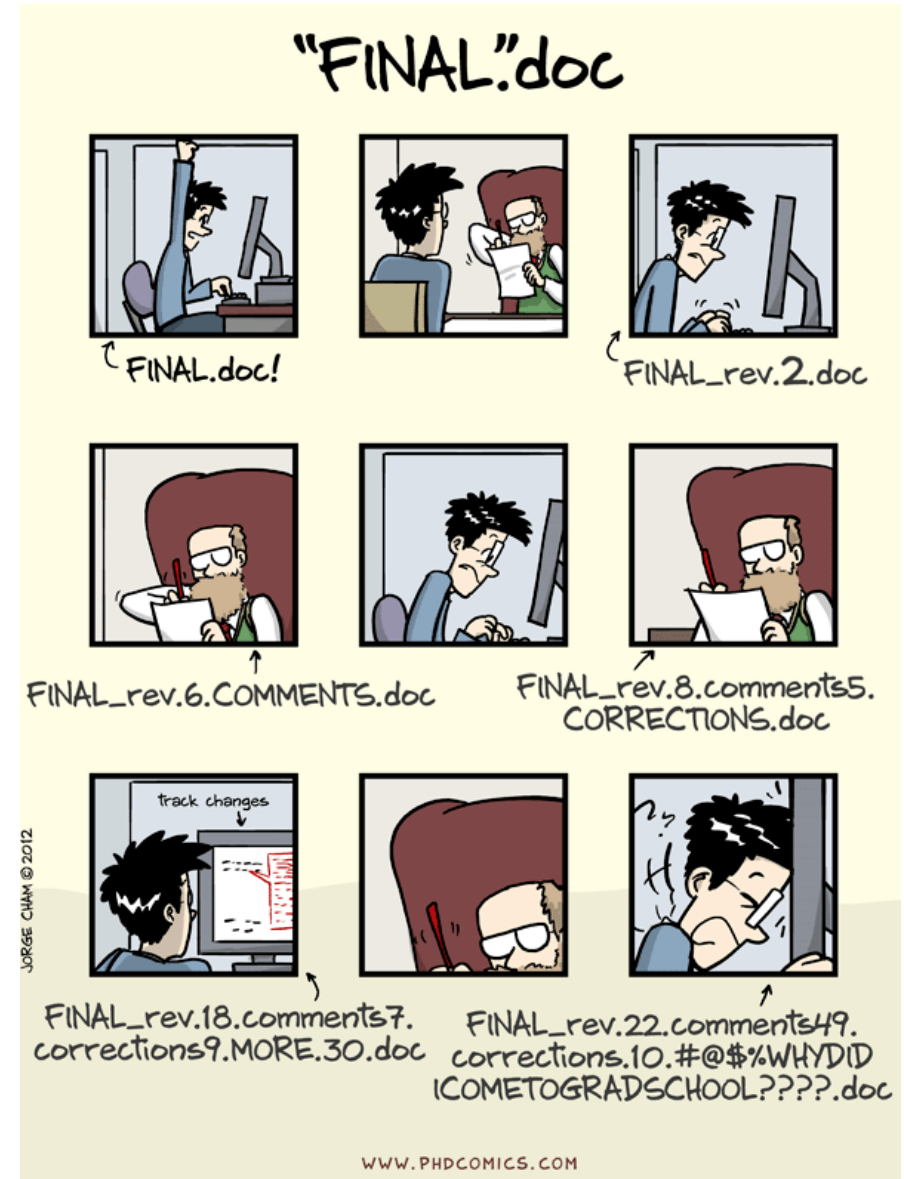
- Intro to Version Control
- Setting up Git
- Creating a Repository
- Tracking Changes
- Exploring History
- More Git Commands
- Managing Conflicts
- Using Git from Rstudio\*

Questions **welcomed** at any point!  
No need to wait!

# Introduction to Automated Version Control

# Lesson Objectives

- Understand the benefits of automated version control
- Understand the basics of how automated version control systems work.

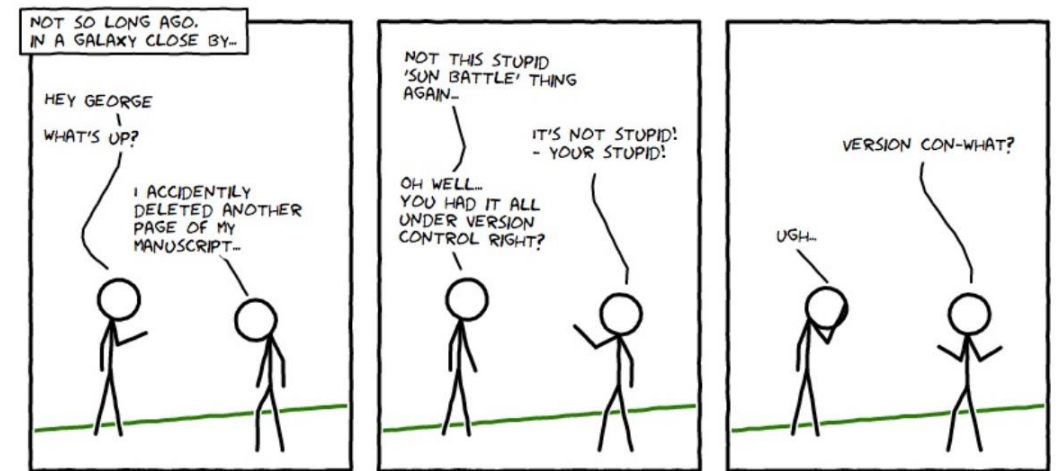


# What is “version control”, and why should you care?

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.

Version Control System (VCS) allows you to revert selected files back to a previous state, revert an 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.

Using a VCS also generally means that if in the updating the document – content or files are lost, it can be easily recover.



<https://smutch.github.io/VersionControlTutorial/>

# Benefits of using version control

Among other reasons, the benefits of using version control are:

- **Collaboration** - Version control allows us to define formalized ways we can work together and share writing and code.
- **Versioning** - Having a robust and rigorous log of changes to a file, without renaming files
- **Rolling Back** - Version control allows us to quickly undo a set of changes. This can be useful when new writing or new additions to code introduce problems.
- **Understanding** - Version control can help you understand how the code or writing came to be, who wrote or contributed particular parts, and who you might ask to help understand it better.
- **Backup** - While not meant to be a backup solution, using version control systems mean that your code and writing can be stored on multiple other computers.

# Objective:

## Understanding the benefits of VCS



Two sets of changes into the same base document

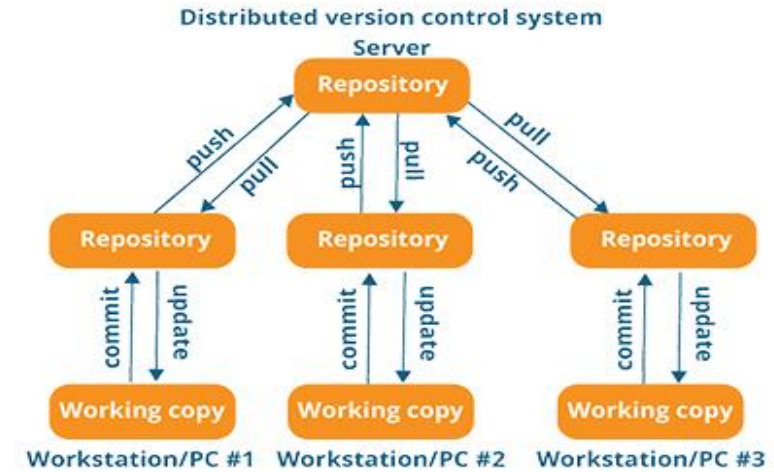
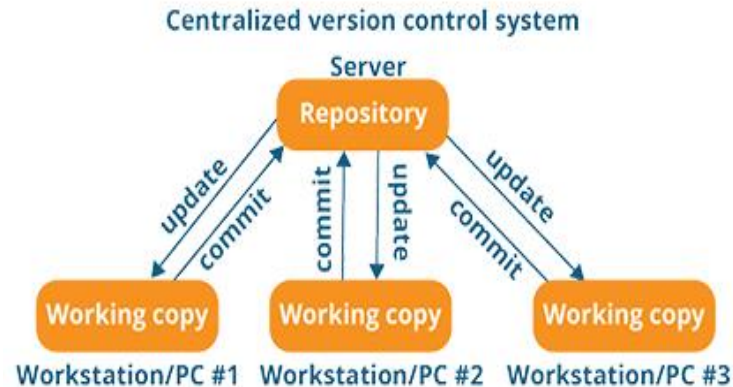


Independent changes on the same document

# Centralized versus Distributed VCS

**Centralized version control systems** are based on the idea that there is a single “central” copy of your project on a remote server and programmers will “commit” their changes to this central copy.

- Microsoft Team Foundation Server [TFS]
- Subversion [SVN]



**Distributed version control systems** do not necessarily rely on a central server to store all the versions of a project's files.

Instead, every person has their own complete copy of the entire repository and make changes as they like. They “commit” changes or “check out” work from the remote repository whenever they like.

A person can also “clone” a copy of a repository and has the full history of the project on their own hard drive. This copy (or “clone”) has all of the metadata of the original.

- Git



# Version Control System: Git

Git is the actual open source distributed version control system [DVCS] that facilitates interactions between a person's local computer and the source code repository.



In Git, every person's working copy of their project is also a repository that can contain the full history of all changes.

*So – what is [GitHub](#), [GitLab](#), [Bitbucket](#), [SVN](#), [CVS](#), [TFS](#) and those other repositories?*

**Git is the tool as well as a distributed system.** These others are remote repositories which can be accessed via the web or on premise repositories are VC **hosting services** or on premise servers **that use Git for source code management.**

Each of these **VCS host providers** offer similar source code tools as they are built on a common tool but **differ** in *price structure, feature sets and ownership.*

# Version Control System:

GitHub vs Bitbucket vs GitLab

**GitHub** has built a community of developers and organizations who contribute to almost every well-known open source libraries, framework or tools.

**VCS Model:** Distributed Version Control System

## Pricing:

- free unlimited private & public-accessible repositories
- Tiered pricing for advanced collaboration support and enterprise customers

Focus on open-source communities



## Ownership:

GitHub was recently acquired by Microsoft causing concern in the open source communities and developers re: intellectual property rights.

*As of 2019, 50 million users and 100 million repositories.*

**Bitbucket** has positioned itself as a professional team option which integrates with other Atlassian collaboration tools e.g. Jira but also provides built-in CI/CD services.

**VCS Model:** Self-Hosted



## Pricing:

- free academic unlimited subscription
- free for small teams up to 5 users w/1GB of large file storage
- standard & premium tiered pricing per month per user

Focus on enterprise and business users

**Ownership:** Atlassian

*April 2019 Atlassian announced that Bitbucket reached 10 million registered users and over 28 million repositories*

# Version Control System:

GitHub vs Bitbucket vs GitLab

**Gitlab** has positioned itself as a professional git repository offering services as containers, orchestration, various reports etc. It's DevOps focus includes tools which can deploy and test applications.

**VCS Model:** Self-Hosted Centralized Version Control System

## Additional services

- Docker Registry / Free Docker registry
- CI tools



## Pricing:

- free and unlimited private and public repositories with limited CI pipelines per month & project issue board

Focus on enterprise and business users

**Ownership:** Open Source | Alphabet Inc. (Google)

*100,000 registered organizations, 30 million registered users and 3000 contributors.*

# Terminology

**Commit** - each record of changes

**Repository** - complete history of commits for a particular project and their metadata

# Summary

- Version control is used to keep track of what one person did and when. It is a system which provides an unlimited 'undo' when working on a document or file.
- Automated version control systems track changes + metadata about the document as separate content from the content of the base (initial version) of a document or file.
- Repositories can be kept in sync across different computers, facilitating collaboration among different people.
- Modern VC systems offer powerful merging tools that make it possible for multiple authors to work on the same files concurrently.

# Setting up Git

# Objective:

## Configure git for the first time

1. Set contributor name and email address
2. Set preferred text editor
3. Best practices for settings to use for every project (global configurations)

In Bash, Git commands begin with the `git` keyword =>

`git verb options`

`verb` => what you want to do

`options` => additional information that might be needed to accomplish what you want to do

# Objective:

## Understand `--global` configuration

The `--global` option flag indicates to use the setting throughout the project. To configure your user name and email address, we will follow Dracula's example except...

Bash

```
$ git config --global user.name "Vlad Dracula"  
$ git config --global user.email "vlad@tran.sylvan.ia"
```

*Use your own GitHub username and email address!*



### Tips & Tricks

To keep your email private:

```
git config -global user.email "<github username>@users.noreply.github.com"
```



# Objective:

## Understand `--global` configuration

```
warning: LF will be replaced by CRLF in .gitignore.  
The file will have its original line endings in your working directory
```

To change the way Git recognizes and encodes line endings:

```
git config --global core.autocrlf true [Windows]
```

```
git config --global core.autocrlf input [MAC]
```



NOTE: `crlf` stands for *carriage return line feed*.

Windows inserts 2 chars and Linux/macOS inserts a new line (one) character.

*Why configure this?* Git will report diff errors when evaluating source code. The various carriage returns will create diff errors.

# Objective: Understand `--global` configuration

To set preferred text editor: `git config --global core.editor "<text editor>"`

Editor	Configuration command	
Atom	<code>\$ git config --global core.editor "atom --wait"</code>	
nano	<code>\$ git config --global core.editor "nano -w"</code>	Universally available Has shortcut help at bottom of window
BBEdit (Mac, with command line)	<code>\$ git config --global core.editor "bbedit"</code>	
Sublime Text (Win, 32-bit install)	<code>\$ git config --global core.editor '"c:/program files (x86)/sublime text 3/sublime_text.exe' -w"</code>	
Sublime Text (Win, 64-bit install)	<code>\$ git config --global core.editor '"c:/program files/sublime text 3/sublime_text.exe' -w"</code>	
Notepad++ (Win, 32-bit install)	<code>\$ git config --global core.editor '"c:/program files (x86)/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"</code>	
Notepad++ (Win, 64-bit install)	<code>\$ git config --global core.editor '"c:/program files/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"</code>	
VS Code	<code>\$ git config --global core.editor "code --wait"</code>	
Emacs	<code>\$ git config --global core.editor "emacs"</code>	
Vim	<code>\$ git config --global core.editor "vim"</code>	Default

# Objective:

## Understand `--global` configuration

If you need to use git via proxy, use the global configuration command:

Bash

```
$ git config --global http.proxy proxy-url  
$ git config --global https.proxy proxy-url
```

To disable the proxy, use the following global configuration command:

Bash

```
$ git config --global --unset http.proxy  
$ git config --global --unset https.proxy
```



To view listing of configurations at anytime: `git config --list`

To access and view the help manual : `git config --help` or `git config -h`

# Summary

There are a few steps that should be completed when first setting up git on your development environment.

Commands start with the `git` keyword followed by a verb plus options. To configure global settings - use: `git config --global`

To configure the programmers (your) username and email address:

```
$ git config --global user.name "Rayvn Manuel"
```

```
$ git config --global user.email "r2c0der@users.noreply.github.com"
```

To configure the preferred editor:

```
$ git config --global user.editor "vim"
```





Creating a local repository

# Scenario

- Four colleagues working at research centers in Maryland, Panama, United Arab Emirates and Alaska.
- Currently the group emails text file to HQ in MD with monthly write-ups about the weather – comparison with last year and oddities.
- Write-ups are copied/pasted into a single file.
- Each site keeps a spreadsheet of weather conditions data e.g. temperature, barometric pressure and precipitation

# Objective: Create a local Git repository

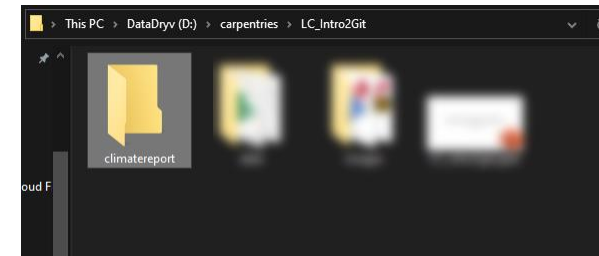
Create a directory to store project files and local repository.

1. Create a folder - `mkdir climaterreport`

```
$ mkdir climaterreport|
```

2. Once its created, move into (change directory) the new directory so we can make create a local repository.

```
$ cd climaterreport/
```



# Objective: Create a local Git repository

3. To create a repository within the planets project folder, we **initialize** it using the command: **git init**

```
$ git init  
Initialized empty Git repository in D:/carpentries/LC_Intro2Git/cli  
matereport/.git/
```

The system will display a message that will verify that the repository and all subdirectories was created or initialized. If you browse to the directory, you will also find a .git subdirectory.



# Objective:

## Describe purpose of .git directory

If you execute the bash `ls` command to list the contents of the planets directory, it'll appear empty.

Adding the `-a` flag will show all hidden files and directories.

```
$ ls -a
./  ../  .git/
```

Adding the `-l` flag will show additional metadata about the directory.

```
$ ls -la
total 8
drwxr-xr-x 1 197610 0 Nov  3 14:22 ./
drwxr-xr-x 1 197610 0 Nov  3 14:16 ../
drwxr-xr-x 1 197610 0 Nov  3 14:22 .git/
```



The .git directory stores all information about your project. Deleting this directory will mean you also delete your projects history.

# Objective:

## Describe how to check project status

To check the status of the project – which branch or whether or not files have been saved (**committed**) to local repository or that the repository has been created:

Use the command: **git status**

```
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

# Objective:

## Describe how to check for an existing .git directory

The **purpose** of .git directory is to create a place to store and track **changes** of a project. As well as have the **ability** to playback or rollback **changes** if necessary.

- The .git directory stores this information and only needs to be created once.
- Creating multiple .git directories defeats this purpose and will cause 'nested' repositories with conflicting tracking histories.
- Before creating a new .git directory, use the **git status** command to check for conflicts:

```
$ git status  
fatal: not a git repository (or any of the parent directories): .git
```

# Objective:

## Describe how to remove files from repository

To remove (rm) files from the working tree of the repository but not your project directory – use bash command `rm`

Bash

```
$ rm filename
```

To remove an entire directory, use bash command `rm -r` (recursive) or the use the flag `-f` to force the execution.

```
$ rm -rf dirName
```

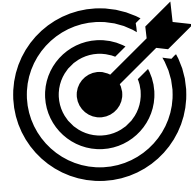


To ensure you do not delete the wrong directory, its always a good practice to check your current working directory by running the bash command: `pwd`

# Summary

After performing the initial setup steps, create a local folder to store your project files.

- Navigate into the folder and run the command `git init` – to create or initialize the repository
- Run the command `ls -a` to view all the files and directories to include the hidden ones
- To check the status of the project use the command `git status`



## Tips & Tricks

Only need to execute the `git init` command once.

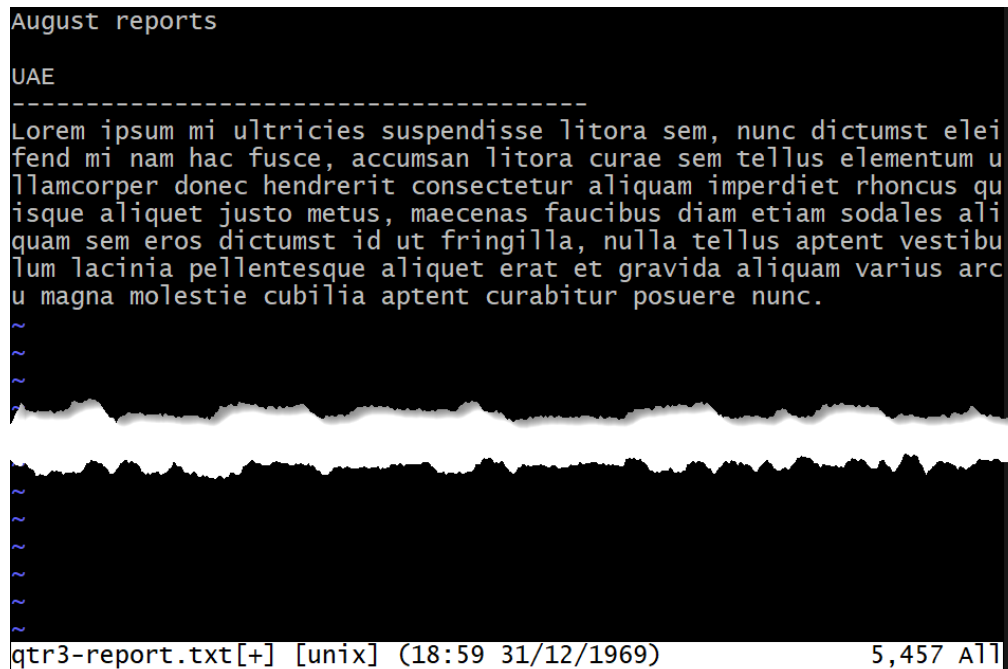
- ✓ This means creating the repository at the highest level folder possible where all necessary project files are found and that you want to share.
- ✓ Avoid adding files and directories to the repository that you do not intend to share or track using git.

# Delving into the Add-Commit Cycle

# Objective:

## Run through **modify** -add-commit cycle

1. Check which directory in which we are working by running the bash command **pwd**
2. Create a new text file **qtr3-reports.txt** and using your favorite text editor add some filler content:



The screenshot shows a terminal window with a text editor open. The editor displays the text "August reports" followed by "UAE" and a separator line. Below the separator is a block of Lorem Ipsum text. The terminal status bar at the bottom shows "qtr3-report.txt[+] [unix] (18:59 31/12/1969) 5,457 All".

<http://www.randomtext.me/>

To begin editing in vim,  
press **i** key for **insert**.

To exit vim, press the **ESC**  
key and type vim  
command **:wq**

# Objective:

## Run through **modify-add-commit** cycle

3. Verify that the file was created and check the contents by using bash commands: **ls** and **cat qtr3-report.txt**

Checking the file status,  
we find that we have an  
untracked changes

```
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    qtr3-report.txt

nothing added to commit but untracked files present (use "git add"
to track)
```

So what is an 'untracked' file? It is a file that git is not aware of and changes for the file are not being versioned (tracked).



# Objective:

## Run through modify-add-commit cycle

4. Type `git add qtr3-report.txt` to add to the project's tracking history.

```
$ git add qtr3-report.txt
warning: LF will be replaced by CRLF in qtr3-report.txt.
The file will have its original line endings in your working directory
```

```
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   qtr3-report.txt
```

Type `git status` to confirm that the file has been staged or being tracked by

# Objective:

## Run through **modify-add-commit** cycle

Git is now tracking the changes but the file has not been saved and a copy has not been stored inside the .git directory.

Use the **git commit** command to accomplish this.

**git commit -m** <message> records a short, descriptive comment.

**git commit** **with no flag** will launch the default configured editor to allow a longer message to be captured.



**Good commit messages** are brief (less than 50 characters) and describe the changes made. "If applied, this commit will....."

Leaving a blank line between the summary line, you can leave additional detailed notes.

# Objective:

## Describe how to use `git log`

Type `git status` to check if everything is up-to-date.

Type the command `git log` to list all commits made to the repository.

The log is a listing:

- In reverse chronological order
- Includes the commit's full identifier which starts with the short identifier displayed after saving/committing the file
- Displays the author/programmer of the commit
- When the commit was created
- The actual message that was added to the commit

```
$ git commit -m "Added the August report for UAE"
[master (root-commit) a3dad4f] Added the August report for UAE
1 file changed, 5 insertions(+)
create mode 100644 qtr3-report.txt
```

```
$ git log
commit a3dad4f6662ee1d4e7f37c3499f9fb4d80ecb5c1 (HEAD -> master)
Author: Rayvn Manuel <manuelr@si.edu>
Date: Tue Nov 3 14:53:26 2020 -0500

    Added the August report for UAE
```

# Objective:

## Run through modify-add-commit cycle

Update the file with new text

Check the status after the update:

`git status` – will confirm the file was modified but report that “no changes added to commit”

Type the command

`git add qtr3-report.txt` or `git add .`

to add changes to working directory.

If all is well, save (commit) the modified file, use the commands:

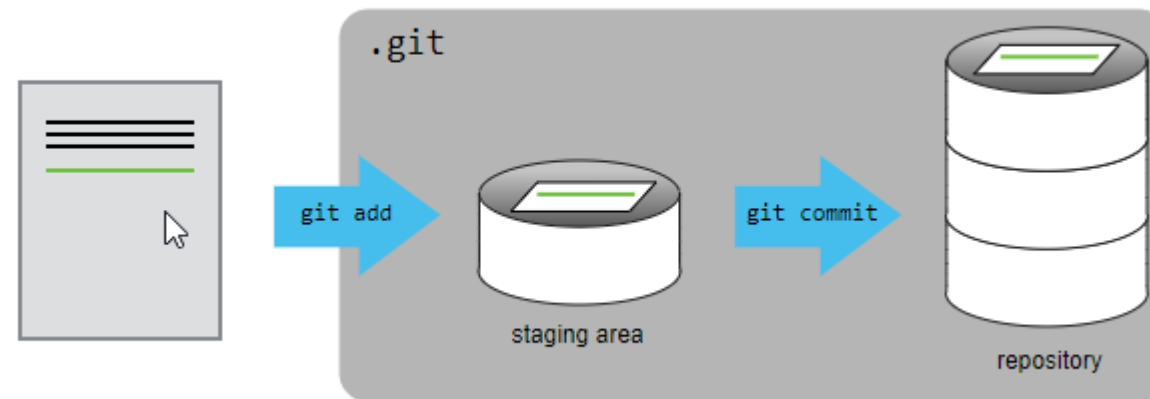
`git commit -m “ <add some relevant comments> ”.`

# Objective:

## Describe the Staging Area

Executing the `git add` command specifies the changes to be tracked and saves the changes to the staging area.

Executing the `git commit` will save the changes and places them in permanent storage (a record is made and stored in the `.git` directory).

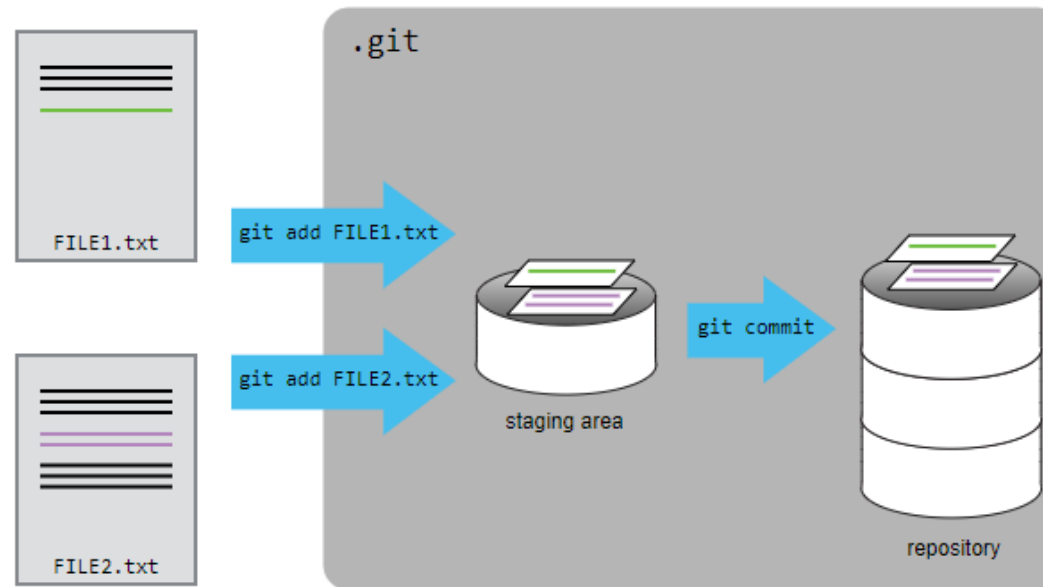


`git diff` will display no output after changes have been tracked but prior to being committed.

# Objective:

## Describe how Git tracks directory

1. Git does not track directories – only the files within them.
2. You can add all files in a directory all at once.



# Additional Notes about the log file

## 🚀 Paging the Log

When the output of `git log` is too long to fit in your screen, `git` uses a program to split it into pages of the size of your screen. When this “pager” is called, you will notice that the last line in your screen is a `:`, instead of your usual prompt.

- To get out of the pager, press `Q`.
- To move to the next page, press `Spacebar`.
- To search for `some_word` in all pages, press `/` and type `some_word`. Navigate through matches pressing `N`.

## 🚀 Limit Log Size

To avoid having `git log` cover your entire terminal screen, you can limit the number of commits that Git lists by using `-N`, where `N` is the number of commits that you want to view. For example, if you only want information from the last commit you can use:

### Bash

```
$ git log -1
```

### Output

```
commit 005937fbe2a98fb83f0ade869025dc2636b4dad5
Author: Vlad Dracula <vlad@tran.sylvan.ia>
Date: Thu Aug 22 10:14:07 2013 -0400
```

```
Discuss concerns about Mars' climate for Mummy
```



# Additional Notes about Directories

## ✈ Directories

Two important facts you should know about directories in Git.

1. Git does not track directories on their own, only files within them. Try it for yourself:

### Bash

```
$ mkdir spaceships  
$ git status  
$ git add spaceships  
$ git status
```

Note, our newly created empty directory `spaceships` does not appear in the list of untracked files even if we explicitly add it (via `git add`) to our repository. This is the reason why you will sometimes see `.gitkeep` files in otherwise empty directories. Unlike `.gitignore`, these files are not special and their sole purpose is to populate a directory so that Git adds it to the repository. In fact, you can name such files anything you like.

2. If you create a directory in your Git repository and populate it with files, you can add all files in the directory at once by:

### Bash

```
git add <directory-with-files>
```

# Exercise

- Create a new repository, add a couple of directories and files.
- Add some content to your files.
- Follow the workflow of adding the changes for tracking and committing the changes to the new repository.

Breakout Room

# Summary

- Files can be stored in a project's working directory (which users see), the staging area (where the next commit is being built up) and the local repository (where commits are permanently recorded).
- Git uses a two-stage commit process. Changes to files must first be added to the staging area, then committed to the repository.
- Write a commit message that accurately describes your changes.



**git status** shows the status of a repository.

**git add** puts changes to a file in the staging area.

**git commit** saves the staged content as a new commit in the local repository.

**git log** lists all commits made to the repository.

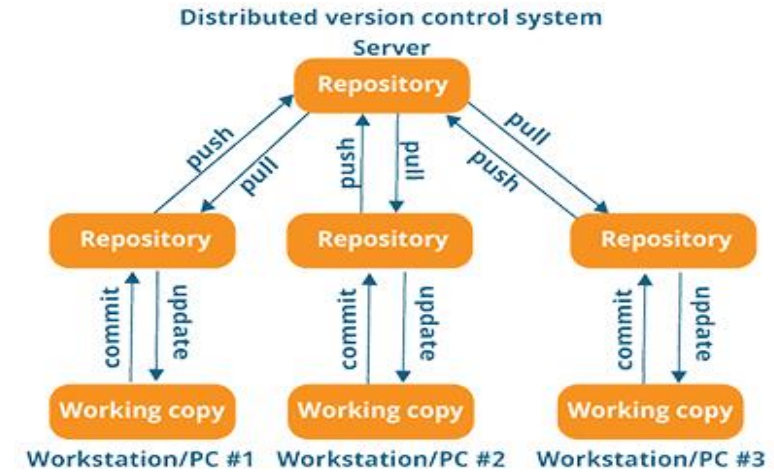
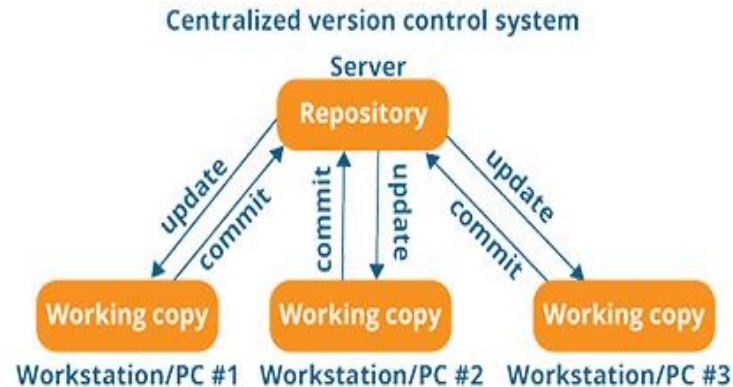
Let's take a break!

# Sharing your work - Remote GitHub Repositories

# Centralized versus Distributed VCS

Centralized version control systems are based on the idea that there is a single “central” copy of your project on a remote server and programmers will “commit” their changes to this central copy.

- Microsoft Team Foundation Server [TFS]
- Subversion [SVN]



Distributed version control systems do not necessarily rely on a central server to store all the versions of a project's files.

Instead, every person has their own complete copy of the entire repository and make changes as they like. They “commit” changes or “check out” work from the remote repository whenever they like.

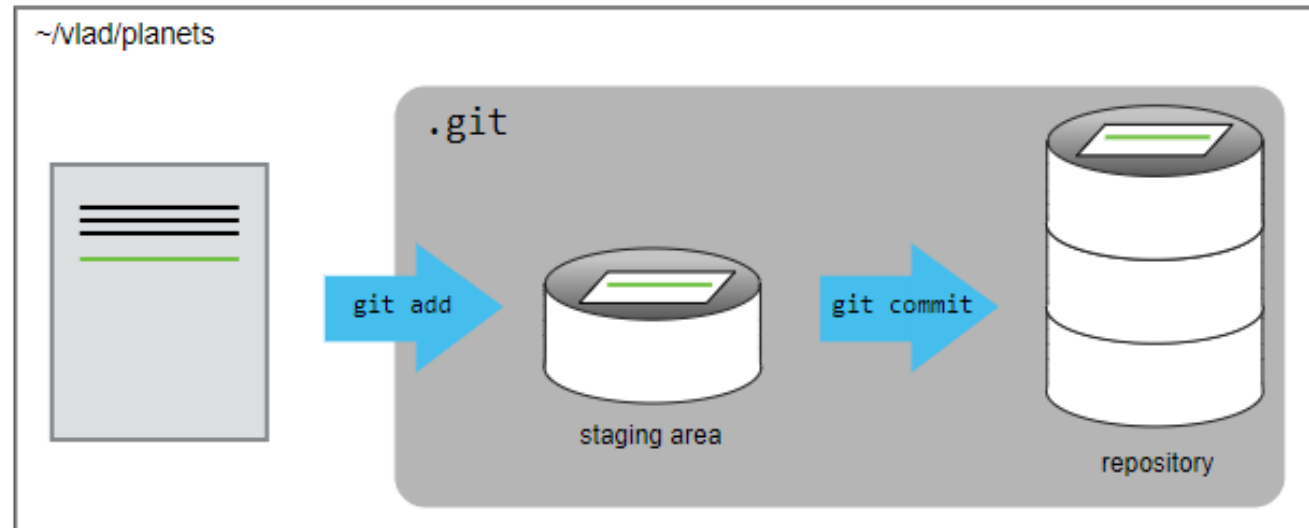
A person can also “clone” a copy of a repository and has the full history of the project on their own hard drive. This copy (or “clone”) has all of the metadata of the original.

- Git

# Objective:

## Explain what remote repositories are

Local git repository



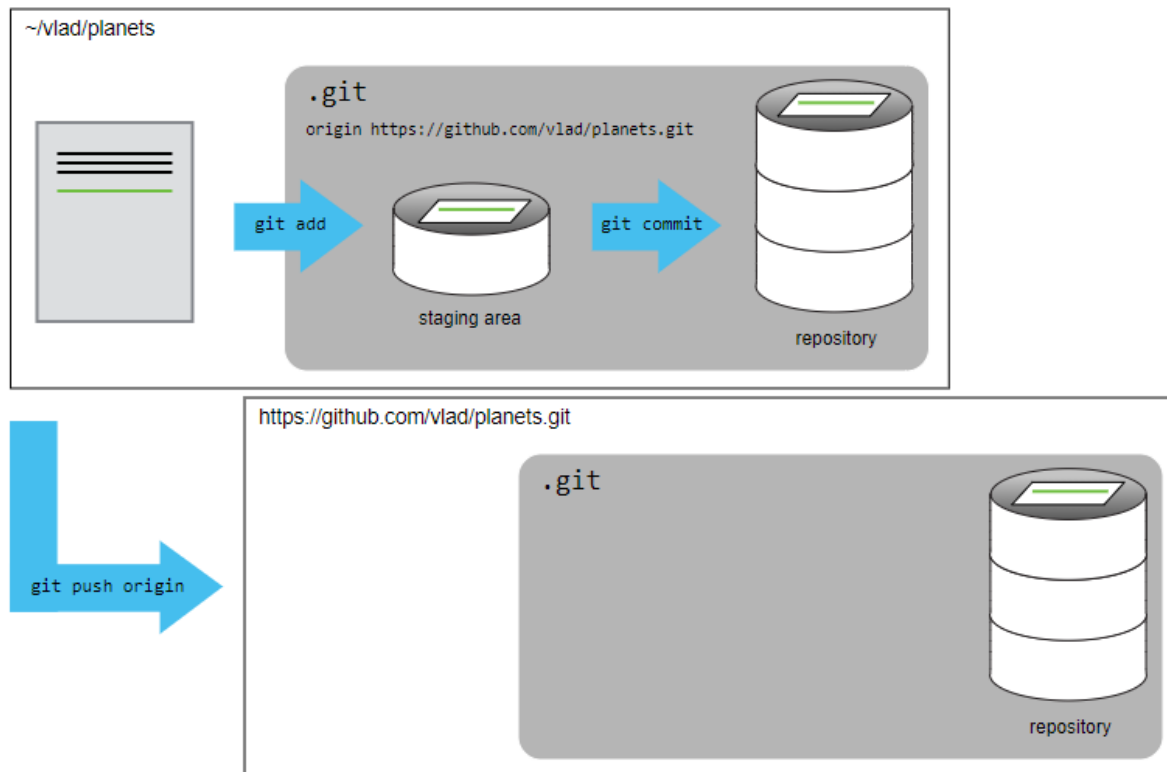
Hosting VC Services  
GitHub, Bitbucket, GitLab

**Commonality:** code collaboration and version control tools offering repository management using Git.



# Objective:

## Push to or pull from a remote repository



### simple daily git workflow





# Other Notes

## ✦ Choosing a license

Choosing a license is an important part of openly sharing your creative work online. For help in wading through the many types of open source licenses, please visit <https://choosealicense.com/>.

## ✦ HTTPS vs. SSH

We use HTTPS here because it does not require additional configuration, which vary from operating system to operating system. If you start using Git regularly, you would like to set up SSH access, which is a bit more secure and convenient, by following one of the great tutorials from [GitHub](#), [Atlassian/BitBucket](#) and [GitLab](#) (this one has a screencast).

## ✦ Why `origin`?

`origin` in the `git remote add` line is just a short name or alias we're giving to that big long repository URL. It could be almost any string we want, but by convention in git, it is usually called `origin`, representing where the repo originated.

## ✦ The README file

It is good practice to add a README file to each project to give a brief overview of what the project is about. If you put your README file in your repository's root directory, GitHub will recognize and automatically surface your README to repository visitors

# Summary

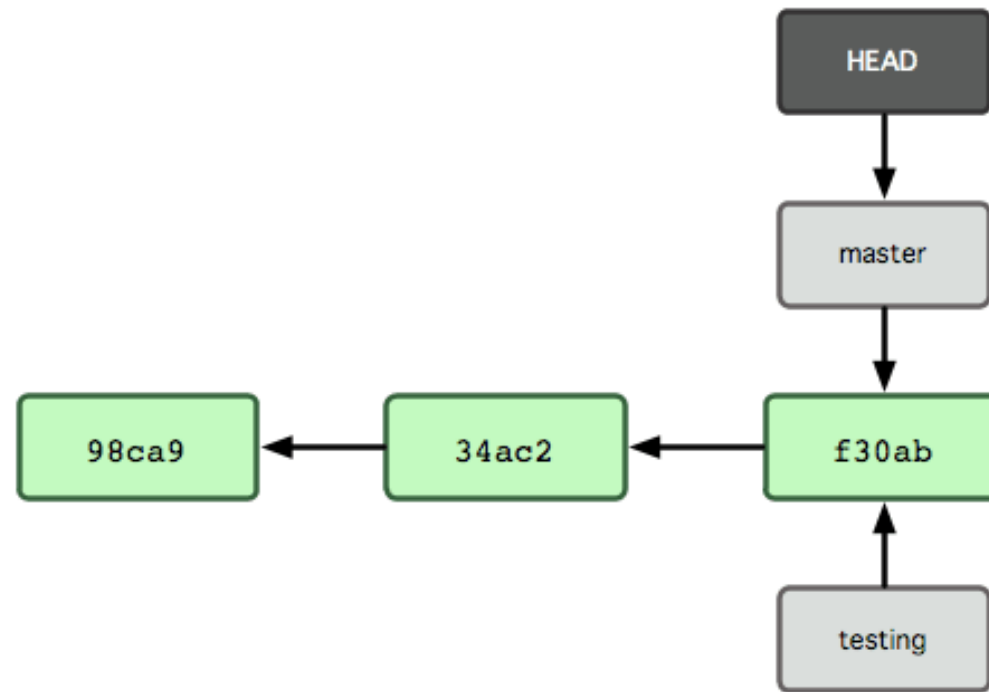
- A local Git repository can be connected to one or more remote repositories.
- Use the HTTPS protocol to connect to remote repositories until you have learned how to set up SSH.
- `git push` copies changes from a local repository to a remote repository.
- `git pull` copies changes from a remote repository to a local repository.

Reviewing history &  
Restoring Version

# Objective:

## Explain what is HEAD in the repo

**HEAD** – refers to the most recent commit of the working directory within the current branch.



# Objective:

## Identify and use Git commit revision number

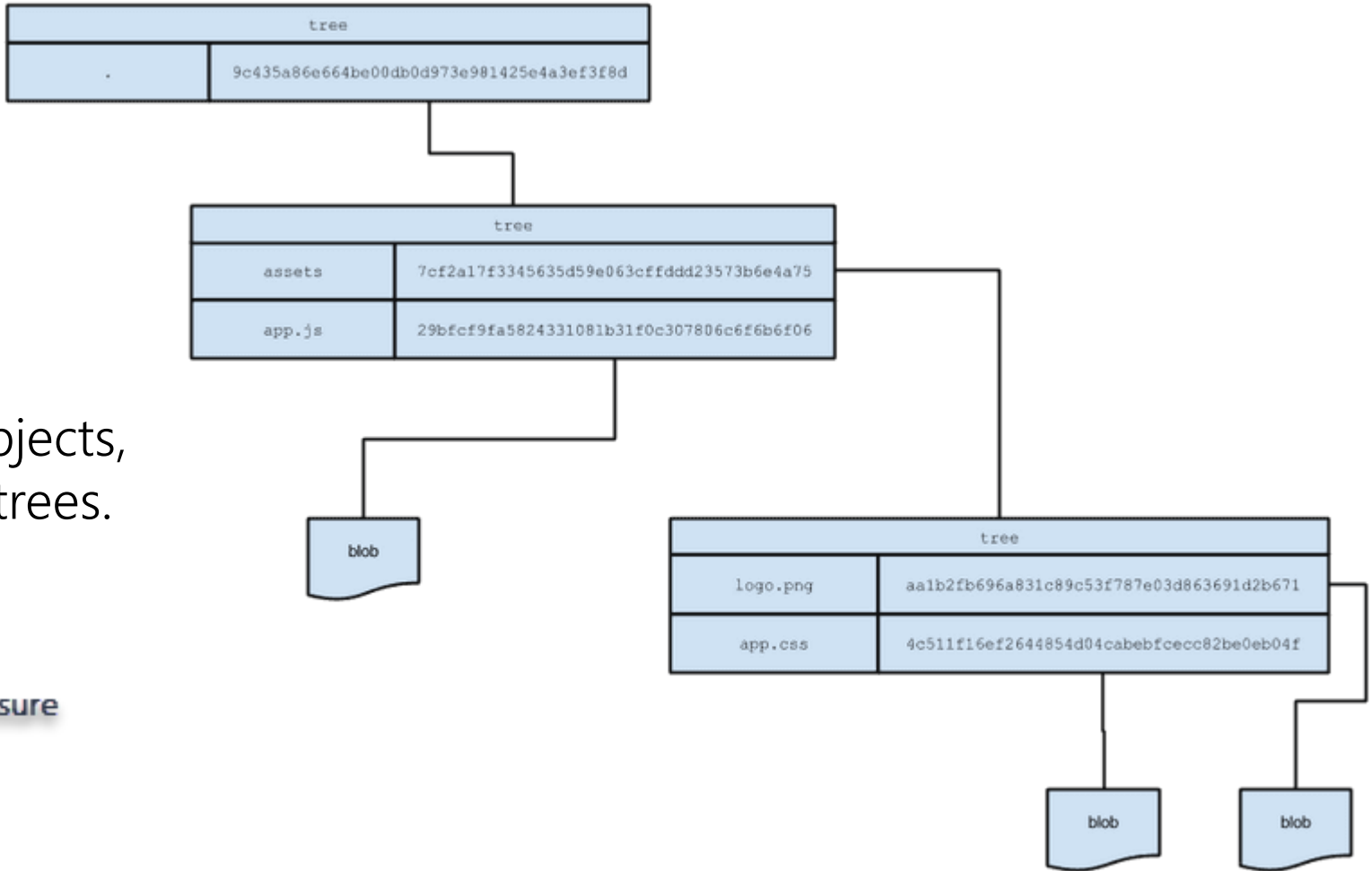
```
.  
├── .git (contents left out)  
├── assets  
│   ├── logo.png  
│   └── app.css  
└── app.js
```

Git's internal storage: commit objects, annotated tag objects, blobs & trees.

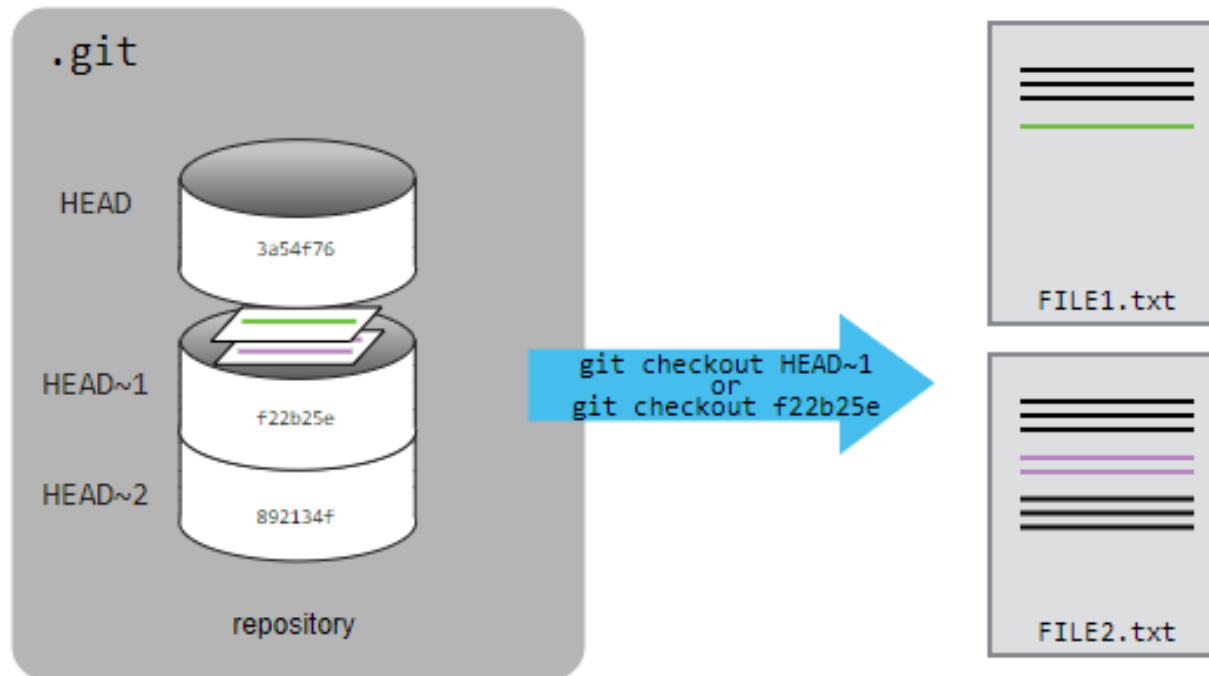
40 character sha1 hash

Creates a unique number which is used to ensure the integrity of the file.

`commit 54fa0ffb82aea83ed3f0e67a103f649d3d14e51a (HEAD -> master)`



# Commit identifiers



Commit identifiers are a reference to the repository at a particular state of change.

In general, when restoring to a previous version – `git checkout HEAD~1` or `HEAD~*` or use the appropriate identifier.

# Objective:

## Compare various versions of tracked file

git diff

**patch** - collection of differences between files and how the differences are applied to the file or source code.

```
$ git diff

diff --git a/mars.txt b/mars.txt  comparing old version a/mars.txt with new version b/mars.txt
index 077071c..42d92e3 100644
--- a/mars.txt
+++ b/mars.txt
@@ -1,2 @@
 Cold and dry, but everything is my favorite color.
+The two moons may be a problem for Wolfman.
```

Which versions of the the file by index

Sections pre and appended with @@ are called 'hunks' . Use contextual line numbers to instruct patch tool (or other processors know where to apply changes.

# Objective:

## Compare various versions of tracked file

```
commit 900c31b8e8340b540c7492925a792505d6ddb7 (HEAD -> master)
Author: Rayvn Manuel <r2coder@aol.com>
Date: Sat Sep 12 17:19:48 2020 -0400
```

Discuss concerns about Mars' climate for Mummy.

```
commit 54fa0ffb82aea83ed3f0e67a103f649d3d14e51a
Author: Rayvn Manuel <r2coder@aol.com>
Date: Sat Sep 12 12:06:55 2020 -0400
```

Add concerns about effects of Mars' moons on Wolfman.

```
commit 5284d5010ea27481aee8c7816c9d7048b45b5da9
Author: Rayvn Manuel <r2coder@aol.com>
Date: Sat Sep 12 11:32:32 2020 -0400
```

Start notes on Mars as a base.

git show

```
$ git show
commit 900c31b8e8340b540c7492925a792505d6ddb7 (HEAD -> master)
Author: Rayvn Manuel <r2coder@aol.com>
Date: Sat Sep 12 17:19:48 2020 -0400
```

commit hash identifier

Discuss concerns about Mars' climate for Mummy.

```
diff --git a/mars.txt b/mars.txt
index 42d92e3..97e54de 100644
```

git file index

--- a/mars.txt

+++ b/mars.txt

@@ -1,2 +1,3 @@

Cold and dry, but everything is my favorite color.

-The two moons may be a problem for Wolfman.

+The two moons may be a problem for Wolfman

+But the Mummy will appreciate the lack of humidity.

changes we made at an older commit as well as the commit message



# Objective:

## Restore old versions of files

`git checkout` restores an old (or current) version of a file

### Bash

```
$ git checkout HEAD mars.txt  
$ cat mars.txt
```

### Output

```
Cold and dry, but everything is my favorite color  
The two moons may be a problem for Wolfman  
But the Mummy will appreciate the lack of humidity
```

### Bash

```
$ git checkout f22b25e mars.txt
```

### Bash

```
$ cat mars.txt
```

### Output

```
Cold and dry, but everything is my favorite color
```

# Detached HEAD



```
$ git checkout 5284d5010e
Note: switching to '5284d5010e'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

    git switch -c <new-branch-name>

Or undo this operation with:

    git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 5284d50 Start notes on Mars as a base.
```

NOTE: Use `git checkout` to reattach (restore) HEAD to the appropriate branch.

# Summary

- `git diff` displays differences between commits and current working directory.
- `git show` displays differences between current and previous commits.
- `git checkout` recovers old versions of files.
- `git revert` reverses changes committed to the local and remote repositories.



Bash command `cat` reads data from a file and outputs the content to the screen. It can be used to create, view and/or concatenate files.

Ignoring Things

# Objective:

## Explain why ignoring files can be useful

```
# Ignore Mac system files
.DS_store

# Ignore node_modules folder
node_modules

# Ignore all text files
*.txt

# Ignore files related to API keys
.env

# Ignore SASS config files
.sass-cache
```

<https://www.freecodecamp.org/news/gitignore-what-is-it-and-how-to-add-to-repo/>

Project folders contain many other directories and files which are related and relevant for the project but do not need to be tracked.

Additionally, integrated development environments (ide) and operating systems create hidden system files which you do not want tracked.

A gitignore file specifies intentionally untracked files that Git should ignore. Files already tracked by Git are not affected.

Additional Resources:

- [Git-SCM: Gitignore](#)
- [gitignore.io](https://gitignore.io)
- [GitHub: Gitignore Templates](#)

# Summary

- The .gitignore file tells Git what files to ignore and not track.
- An exclamation point (!) operator will create an exception/exclusion in .gitignore file.
- The order of operations or application of rules is read from the top down.



## NOTE FROM Tracking Changes:

- Git does not track directories on their own, only files within them.
- **.gitkeep** files are created and stored in empty directories in order to force git to track the directory

\* is used as a wildcard match



/ is used to ignore pathnames relative to the .gitignore file

# is used to add comments to a .gitignore file

Review

# Exercise

- Create a new remote repository
- Add the remote repository to a new local repository
- Add directories and files with dummy content
- Add a .gitignore file and configure it to ignore certain files in your local setup
- Follow the workflow of adding the changes for tracking and committing along with pushing and pulling files to and from the local and remote repositories.

Breakout Room



# Version Control with Git