Command-Line Proficiency:
Git and GitHub

Learning Objectives:

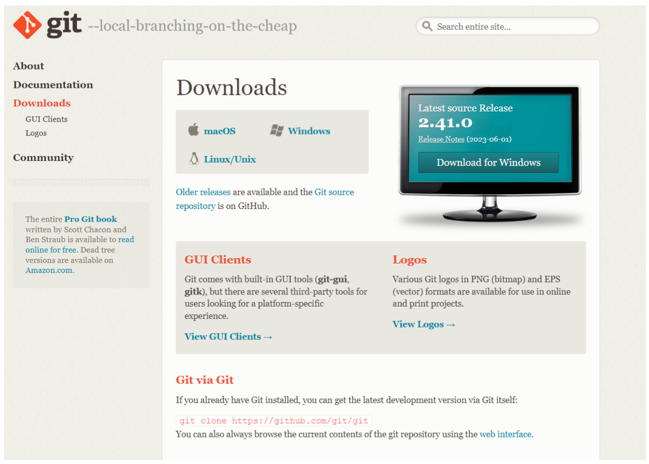
- Understand the fundamentals of version control systems and the benefits of using Git.
- Learn how to initialize a Git repository and configure Git on your local machine.
- Master basic Git commands for tracking changes, committing, and managing versions.
- Gain proficiency in branching and merging strategies to effectively manage project development.
- Learn how to collaborate with remote repositories and GitHub for seamless teamwork.
- Understand the concept of forking and its significance in contributing to open-source projects.

Warm Up:

1. Open git
2. Check the version of your git using Git bash.
3. Go to your Desktop folder
4. Create a folder and name it 'Git'

Installation of Git

1. Download and install Git from the official website (https://git-scm.com/downloads) based on your operating system (*Windows, macOS, Linux*).



The entire **Pro Git book** written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

Downloads

Release Notes (2023-06-01)

Older releases are available and the Git source repository is on GitHub.

GUI Clients

Git comes with built-in GUI tools (**git-gui**, **gitk**), but there are several third-party tools for users looking for a platform-specific experience.

Logos

Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.

Git via Git

If you already have Git installed, you can get the latest development version via Git itself:

```
git clone https://github.com/git/git
```

You can also always browse the current contents of the git repository using the web interface.

2. Open a terminal or command prompt to verify the installation by running the following command:

```
git --version
```

Lesson Proper

Git is a distributed version control system that allows multiple developers to work collaboratively on a project. It tracks changes to files and directories over time, enabling easy collaboration, branching, merging, and more.

Step 1: Configuring Git

Configure Git with your username and email by running the following commands, replacing the placeholders with your own information:

```
git config --global user.name "Your Name"
git config --global user.email "your-email@example.com"
```

What is the **global** flag?

Global Git config controls settings for the currently logged in user and all his repositories.

Other options, `git config -h` or `git help config` (opens a manual of commands)

Step 2: Initializing a Git Repository

1. Create a new directory for your project named “**Git-Practice**” and navigate to it in the terminal or command prompt.
2. Run the following command to initialize a new Git repository.

```
git init
```

This command creates a new empty Git repository in the current directory.

Step 3: Working with Git Locally

1. Create a new file named "**file1.txt**" in the project directory.
2. Open the "file1.txt" file in a text editor and add some content.
3. Run the following command to check the status of your repository:

```
git status
```

This command shows the status of your files, highlighting any changes.

4. Stage the "file1.txt" file for commit by running the following command:

```
git add file1.txt
```

This command adds the "file1.txt" file to the staging area, indicating that it will be included in the next commit.

5. Commit the changes with a descriptive message by running the following command:

```
git commit -m "Added file1.txt file"
```

This command creates a new commit with the changes made to the "file1.txt" file.

6. Create another file named "file2.txt" and make some changes.
7. Stage and commit the "file2.txt" file using the same commands as before.

Step 4: Branching and Merging

1. Create a new branch named "feature" by running the following command. This command creates a new branch without switching to it.

```
git branch feature
```

2. Switch to the "feature" branch by running the following command:

```
git checkout feature
```

This command switches your working directory to the "feature" branch.

3. Make some changes to the "file1.txt" file in the "feature" branch. Create a new file named "file3.txt" and add some code to it.
4. Stage and commit the changes in the "feature" branch using the same commands as before.
5. Switch back to the main branch by running the following command:

```
git checkout main
```

This command switches your working directory back to the main branch.
** Note: **main** can sometimes be named **master**.*
If this is the case, use master in the command instead of main

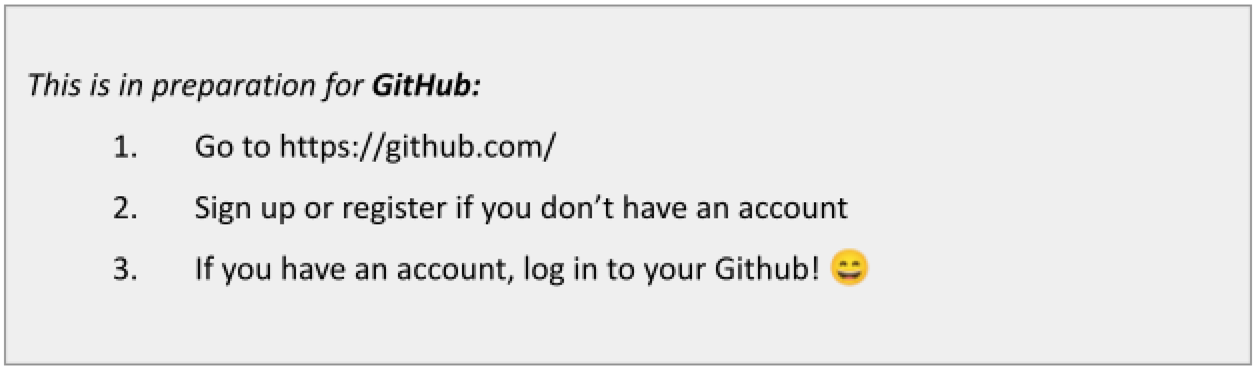
6. Merge the changes from the "feature" branch into the main branch by running the following command:

```
git merge feature
```

This command incorporates the changes made in the "feature" branch into the main branch

7. Create another branch named "bug-fix" and make some changes to the "file1.txt" file.
8. Stage and commit the changes in the "bug-fix" branch.
9. Switch back to the main branch and merge the "bug-fix" branch into it.

What you need to do:


*This is in preparation for **GitHub:***

1. Go to https://github.com/
2. Sign up or register if you don't have an account
3. If you have an account, log in to your Github! 😄

Step 5: Collaborating with Remote Repositories and GitHub

Collaboration is an essential aspect of software development, and Git provides powerful features for working with remote repositories, such as **GitHub**.

GitHub is a web-based platform that helps developers collaborate on software projects. It serves as a repository for code, allowing individuals and teams to manage and track changes to their codebase.

In this step, we will cover cloning, pulling, and forking repositories, which are fundamental to collaborating with others.

1. Create a new repository on GitHub and copy the repository URL.
2. Add the remote repository as the origin by running the following command, replacing with the actual URL:

```
git remote add origin <repository-url>
```

This command adds a remote named "origin" pointing to the GitHub repository.

3. Push your local repository to the remote repository by running the following command:

```
git push -u origin main
```

This command pushes your main branch to the origin remote repository on GitHub.

4. Create a README.md file in your local repository.
5. Stage and commit the README.md file
6. Push the changes to the remote repository.

Step 6: More on Collaborating with Remote Repo

1. Cloning a Remote Repository

Cloning allows you to create a local copy of a remote repository on your machine, enabling you to work on it and contribute changes. To clone a remote repository, use the following command:

```
git clone <repository-url>
```

This command creates a local copy of the remote repository in a new directory with the same name as the repository.

* *Note: **Replace** with the actual **URL** of the remote repository.*
For example: git clone https://github.com/username/repository.git

2. Pulling Changes from a Remote Repository

Pulling changes from a remote repository is a crucial aspect of collaborative software development. It allows you to update your local copy of a repository with changes made by others. Use the following command:

```
git pull origin main
```

This command retrieves the latest changes from the remote repository and merges them into your local repository.

If there are conflicting changes between your local repository and the remote repository, Git will prompt you to resolve the conflicts before merging the changes.

3. Forking a Repository

Forking allows you to create a personal copy of someone else's repository under your own GitHub account. Forking is typically used when you want to contribute to a project hosted on GitHub, but you don't have direct write access to the original repository.

To fork a repository, follow these steps:

- **a.** Visit the repository on GitHub that you want to fork.
- **b.** Click on the "Fork" button on the top-right corner of the repository page. This action creates a copy of the repository under your GitHub account.
- **c.** After forking, you can clone the forked repository to your local machine, make changes, and submit pull requests to the original repository to propose your changes.

ADDITIONAL NOTES : **Mastering the Terminal**

Introduction to the Terminal Environment

- Understanding the terminal interface
- Navigating directories and file structures
- Basic commands: **cd, ls, pwd, mkdir, touch, rm**

```
# Change directory
cd /path/to/directory

# List files and directories
ls

# Print current working directory
pwd

# Create a new directory
mkdir new_directory

# Create a new file
touch new_file.txt

# Remove a file
rm file.txt
```

Helper Commands

- clear – to clear the terminal
- tab – to autocomplete the line
- ↑ and ↓ – to cycle through previous commands

File and Directory Management

- Creating, renaming, and deleting directories and files
- Copying, moving, and linking files and directories
- Working with file permissions: **chmod, chown**

```
# Create a directory
mkdir directory_name

# Rename a file or directory
mv old_name new_name

# Copy a file or directory
cp source destination

# Move a file or directory
mv source destination

# Create a symbolic link
ln -s target_file link_name

# Change file permissions
chmod 755 file_name

# Change file ownership
chown user_name file_name
```

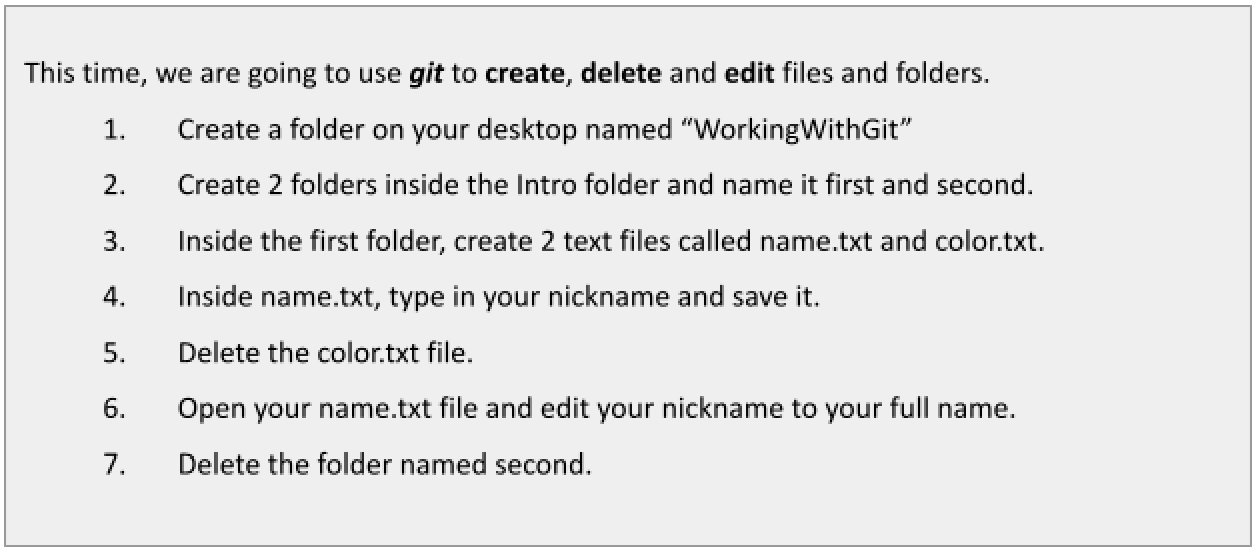
Text Editing in the Terminal

- Using basic text editors: **nano, vim** *(git bash)*
- Creating and editing files
- Navigating within a file
- Saving and exiting the editor

```
# Create and edit a file using nano
nano file.txt

# Create and edit a file using vim
vim file.txt

# Navigating within a file (vim)
j - move down
k - move up
gg - go to the beginning of the file
G - go to the end of the file

# Saving and exiting the editor (vim)
```

```
:w - save changes
:q - quit without saving
:wq - save changes and quit
```

PRACTICE:


This time, we are going to use ***git*** to **create**, **delete** and **edit** files and folders.

1. Create a folder on your desktop named "WorkingWithGit"
2. Create 2 folders inside the Intro folder and name it first and second.
3. Inside the first folder, create 2 text files called name.txt and color.txt.
4. Inside name.txt, type in your nickname and save it.
5. Delete the color.txt file.
6. Open your name.txt file and edit your nickname to your full name.
7. Delete the folder named second.

Working with Processes

- Managing processes: **ps, kill, top**
- Running processes in the background: **&**
- Monitoring process activity: **htop, watch**

```
# List running processes
ps

# Kill a process using its PID
kill PID

# Kill a process by name
killall process_name

# Run a process in the background
command &
```

```
# Monitor system activity in real-time
top

# Monitor process activity in real-time
htop

# Repeat a command periodically
watch -n 5 command
```

File Searching and Manipulation

- Searching for files and directories: **find, grep**
- Bulk renaming files: **rename**
- Text manipulation using regular expressions: **sed, awk**

```
# Search for files by name
find /path/to/directory -name "file.txt"

# Search for files containing specific text
grep "search_text" file.txt

# Rename multiple files using regular expressions
rename 's/old_name/new_name/' *.txt

# Replace text in a file using sed
sed -i 's/old_text/new_text/g' file.txt

# Extract specific columns from a text file using awk
awk '{print $1, $3}' file.txt
```

Package Management

- Installing and updating software packages: **package managers** *(e.g., apt, yum, brew)*
- Searching for packages: **apt search, yum search, brew search**
- Removing packages: **apt remove, yum remove, brew uninstall**

```
# Install a package using package manager
apt install package_name
yum install package_name
brew install package_name

# Update all installed packages
apt update
apt upgrade
yum update
yum upgrade
brew update
brew upgrade

# Search for a package
apt search package_name
yum search package_name
brew search package_name

# Remove a package
apt remove package_name
yum remove package_name
brew uninstall package_name
```

Environment Configuration

- Environment variables: **export, env**
- Shell configuration files: **.bashrc, .bash_profile**
- Modifying **PATH** variable

```
# Set an environment variable
export VARIABLE_NAME=value

# View all environment variables
env

# Edit the .bashrc file
nano ~/.bashrc

# Edit the .bash_profile file
nano ~/.bash_profile
```

```
# Modify the PATH variable
export PATH=$PATH:/new/directory
```

Process Automation with Shell Scripting

- Introduction to shell scripting: **#!/bin/bash**
- Writing and executing shell scripts
- Passing command-line arguments to scripts
- Automating repetitive tasks

```
#!/bin/bash

# Print a message
echo "Hello, world!"

# Read user input
read -p "Enter your name: " name
echo "Hello, $name!"

# Accept command-line arguments
echo "Argument 1: $1"
echo "Argument 2: $2"

# Looping and conditionals
for i in {1..5}; do
    echo $i
done

if [ $1 -gt 10 ]; then
    echo "Greater than 10"
else
    echo "Less than or equal to 10"
```

CLI Command Reference Websites:

60 Essential Linux Commands + Free Cheat Sheet

Mac Terminal Commands {Cheat Sheet With Examples}

ADDITIONAL NOTES: **Setting Up Visual Studio Code**

Visual Studio Code (VS Code) is a free and open-source code editor developed by Microsoft. It is available for Windows, macOS, and Linux.

Installation

1. Go to the Visual Studio Code website: https://code.visualstudio.com/ .
2. Select your operating system and click **Download**.
3. Double-click the downloaded file to install Visual Studio Code.
4. Follow the on-screen instructions to complete the installation.

Navigation

Visual Studio Code is a powerful code editor with a wide range of features. Here is a brief overview of the main navigation areas:

- **Activity Bar:** The activity bar on the left-hand side of the window provides quick access to common tasks, such as opening files, searching for code, and debugging.
- **Explorer:** The explorer displays the files and folders in your project. Use it to navigate and open files.
- **Editor:** The editor is where you write and edit your code. It also includes features such as syntax highlighting, code completion, and error checking.
- **Extensions**: Install and manage extensions.
- **Command Palette**: The command palette is a searchable list of all the commands available in Visual Studio Code. To open it, press Ctrl+Shift+P (or *Cmd+Shift+P on Mac*).

Getting Started

Once you have installed Visual Studio Code, you can create a new text file by clicking the:
File menu and selecting **New > File**. Then, save the file with a **.txt** extension.

To start writing code, simply type into the editor window. Visual Studio Code will provide syntax highlighting and code completion to help you write your code correctly.

Using Terminal:

Click on "**View**" in the top menu, then select "**Terminal**" to open the integrated terminal.

Opening a File:

To open a file in VS Code, click the **File** menu and select **Open File**. You can also drag and drop files into the editor window.’