INTRODUCTION TO GIT

WHAT IS VERSION CONTROL?

WHAT IS VERSION CONTROL?

- A system to manage changes to files over time.
- Enables tracking, collaboration, and versioning.
- Prevents data loss and streamlines teamwork.

Think of Git as a time machine for your projects!

WHY USE GIT ON LINUX?

- Seamless integration with Ubuntu Linux.
- Command-line tools for efficient workflow.
- Flexible configuration for development environments.
- Direct compatibility with popular Linux package managers.

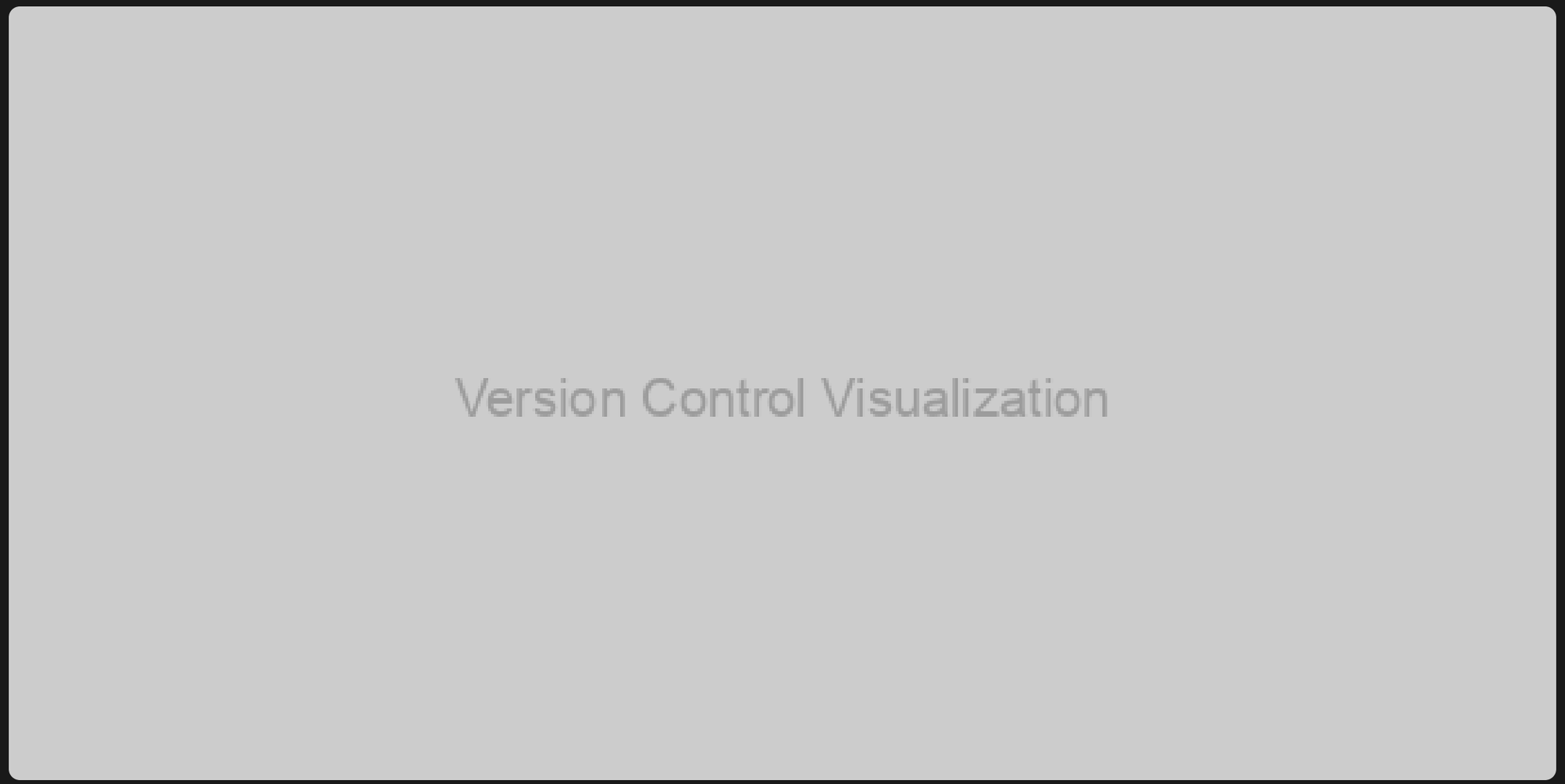
EXAMPLE SCENARIO

- Multiple developers collaborating on a techy story titled "Git-tastic Chronicles."
- Resolve conflicts when multiple edits overlap.
- Use Git to merge contributions seamlessly.

BENEFITS OF VERSION CONTROL

1. Maintain a history of changes.
2. Collaborate with a team effectively.
3. Experiment and roll back safely.
4. Avoid overwriting or losing work.

VISUALIZING VERSION CONTROL



SETTING UP GIT

PREPARING YOUR ENVIRONMENT

STEP 1: INSTALLING GIT

COMMAND:

```
sudo apt update
sudo apt install git
```

EXPLANATION:

- `sudo apt update`: Updates the local package index to ensure you get the latest version.
- `sudo apt install git`: Downloads and installs Git from the Ubuntu repositories.

STEP 2: VERIFYING INSTALLATION

COMMAND:

```
git --version
```

EXAMPLE OUTPUT:

```
git version 2.x.x
```

STEP 3: CONFIGURING GIT

COMMANDS:

```
# Set gedit as the default editor for Git
sudo apt install gedit -y
git config --global core.editor \"gedit\"
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

EXPLANATION:

- `--global`: Applies these settings for all repositories on this machine.
- Associates commits with your name and email for tracking changes.

BASIC GIT OPERATIONS

REPOSITORY-LEVEL OPERATIONS

INITIALIZING VS CLONING A REPOSITORY

OPTION 1: INITIALIZING A NEW REPOSITORY

- Use when starting a project from scratch:

```
git init git-tastic-chronicles
```

Example Output:

```
Initialized empty Git repository in /path/git-tastic-chronicles/
```

OPTION 2: CLONING AN EXISTING REPOSITORY

- Use when collaborating on an existing project:

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

Example:

```
git clone https://github.com/example/git-tastic-chronicle
```

INITIAL PROJECT SETUP

1. Create a new file:

```
gedit README.md # Open the file in gedit and add the content mar
```

2. Stage the file:

```
git add README.md
```

3. Commit the change:

```
git commit -m "Initial commit with README.md"
```

4. Push to remote (if applicable):

```
git push origin main
```

BRANCHING IN GIT

CREATING AND SWITCHING BRANCHES

COMMANDS:

1. Create a new branch for Chapter One edits:

```
git branch cool-feature
```

2. Switch to the branch:

```
git checkout cool-feature
```

3. Shortcut for creating and switching:

```
git switch -c cool-feature
```

EDITING THE FILE IN THE NEW BRANCH

1. Add content to `README.md`:

```
nano README.md # Open the file and add "## Chapter One: The Beginning of Git-tastic Chronicle
```

2. Stage and commit the changes:

```
git add README.md
git commit -m "Add Chapter One to README.md"
```

MERGING BRANCHES

COMMANDS:

1. Switch to the target branch:

```
git checkout main
```

2. Merge the feature branch:

```
git merge cool-feature
```

RESOLVING MERGE CONFLICTS

STEP 1: ATTEMPT TO MERGE

COMMAND:

```
git merge cool-feature
```

EXPLANATION:

This attempts to integrate the changes from `cool-feature` into the current branch. If there are conflicting changes, Git will pause the merge and show the conflict.

STEP 2: RESOLVING THE CONFLICT

SCENARIO:

Two contributors edited the same line in `README.md`, leading to a conflict.
Open the conflicting file to see conflict markers:

EXAMPLE CONFLICT MARKERS:

```
<<<<<<< HEAD
Introduction edits by the main branch
=======
Additions to Chapter One
>>>>>>> cool-feature
```

RESOLUTION:

Edit the file to combine or choose the appropriate changes.

STEP 3: STAGING THE RESOLVED FILE

COMMAND:

```
git add README.md
```

EXPLANATION:

Staging the file marks it as resolved and ready for the next commit.

STEP 4: COMPLETING THE MERGE

COMMAND:

```
git commit -m "Resolve merge conflict in README.md"
```

EXPLANATION:

This finalizes the merge by committing the resolved changes to the repository.

WORKING WITH REMOTES

ADDING AND MANAGING REMOTE REPOSITORIES

ADDING A REMOTE REPOSITORY

COMMANDS:

1. Add a remote repository:

```
git remote add origin <repository-ur
```

2. Push changes to the remote:

```
git push origin main
```

PULLING CHANGES

COMMANDS:

1. Fetch changes:

```
git fetch origin
```

2. Merge fetched changes:

```
git pull origin main
```

KEY TAKEAWAYS

- Repository-level operations set up your project.
- Branch-level operations isolate and manage changes effectively.
- Merge conflicts are resolved manually by editing files.
- Remote repositories enable team collaboration.