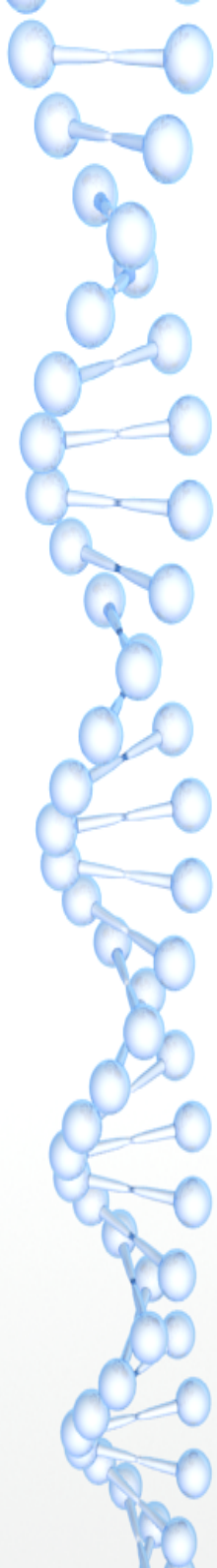


Introduction To Git

Session 1: Git Basics

Outline

- Installing Git
- Creating Your Git Identity
- Getting Help On Git Commands
- Create a Git repository
- Clone a Git repository
- Saving Changes
- View Project History
- Managing Branches

Installing Git

- Installation:
- Debian/Ubuntu (apt-get):

```
$ sudo apt-get update
$ sudo apt-get install git
```

- Fedora (dnf/yum):

```
$ sudo dnf install git
```

OR

```
$ sudo yum install git
```

- Verify installation:

```
$ git --version
```

Creating Your Git Identity

- Every committer is identified by name + email address combination
- These details are associated with every commit you make
- Configure your Git name and email using the following commands, , replacing Brown's details with your own:

```
$ git config --global user.name "Brown Msiska"
$ git config --global user.email "bmsiska@gmail.com"
```

Getting Help On Git Commands

- You can get documentation for a command such as **git log** with:

  $ man git-log

  OR

  $ git help log

Create a Git Repository

- The **git init** command creates a new Git repository
- It can be used to convert an existing, unversioned project to a Git repository
- Or initialize a new empty repository
- To turn a current directory into a Git repo:

```
git init
```

- To create a new empty repo:

```
git init <directory>
```

Clone a Git Repository

- The **git clone** command copies an existing Git repository.
- To clone a repository located at **<repo>** into current directory:

```
git clone <repo>
```

- To clone a repository located at **<repo>** into a specific directory:

```
git clone <repo> <directory>
```

- The original repository can be located on the local filesystem or on a remote machine accessible via HTTP or SSH.

Saving Changes

- The **git add** and **git commit** commands are fundamental to the Git workflow
- They are the means to record changes into the repository.
- Developing a project revolves around the basic **edit/stage/commit** pattern
- you edit your files in the working directory
- Then you stage changes with **git add**
- Then you commit it to the project history with **git commit**
- In conjunction with these commands, you'll also need **git status** to view the state of the working directory and the staging area

Git Add Command

- The **git add** command adds a change in the working directory to the staging area
- It tells Git what changes to include in the next commit.
- To add all changes in a file:

  ```
  git add <file>
  ```

- To add all changes in a directory:

  ```
  git add <directory>
  ```

- To add all changes in the working directory:

  ```
  git add .
  ```

Git Commit Command

- The **git commit** command commits the staged snapshot to the project history
- To commit staged changes:

  ```
  git commit
  ```

- This prompts you for a **commit message**
- To commit with a message:

  ```
  git commit -m "<message>"
  ```

- Alternatively, you can use:

  ```
  git commit -a
  ```

- To commit changes to all tracked (those **added** before) without running git add beforehand

View Repository History

- At any point you can view the history of your changes using:

  ```
  git log
  ```

- Or

  ```
  git log --oneline
  ```

Managing Branches

- A single Git repository can maintain multiple branches of development
- To create a new branch named "experimental", use:

  git branch experimental
- To list branches in your repository:

  git branch
- The current branch will be preceded by an asterisk
- To switch to a specific branch:

  git checkout <branch>
- To merge with changes in another branch:

  git merge <branch-to-merge-with>

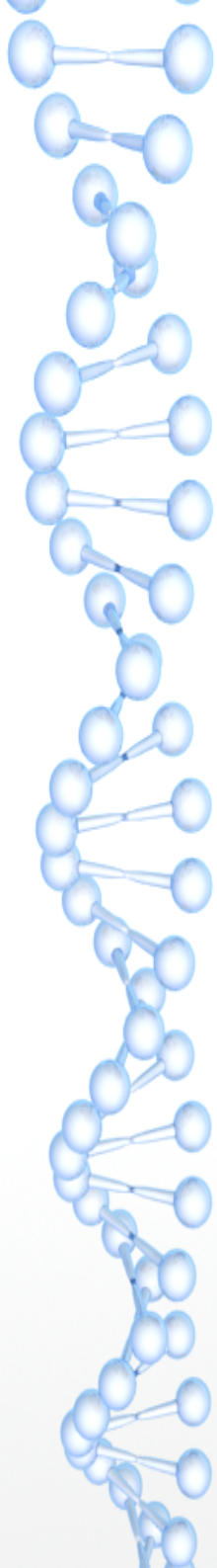
Managing Branches

- To delete a branch use:

  git branch -d experimental

- This will produce an error if the branch has not been merged
- To force delete (for a failed experiment, for example):

  git branch -D experimental



Practice + Break Time