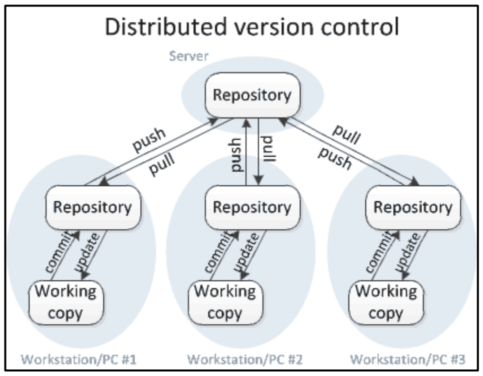
DATA-0200 Lab 12: Introduction to GitHub

"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.corrections9.MORE.30.doc
FINAL_rev.22.comments49.corrections.10.#@$%WHYDID ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

What is Version Control?

- Version control is a record of who makes changes to what and when they did it.
- We can always undo
- Easier to collaborate without overwriting



[1]

Git, specifically, is a distributed version control system, which means that each developer working on a project has their own local copy of the entire project history. This decentralization allows developers to work independently and efficiently and then merge their changes with others when necessary.

Popular platforms like **GitHub**, **GitLab**, and **Bitbucket** host Git repositories, providing a centralized location for collaboration, issue tracking, and other features that enhance the development process.

[1] Source: https://homes.cs.washington.edu/~mernst/advice/version-control.html

Getting Git

You can use a terminal in Mac and a command prompt in Windows. Another preferred way for Windows is to use git bash, which can be downloaded from here: https://git-scm.com/downloads

Setup

First thing to do is to setup your identity. This identifies you to other people who download the project.

```
$ git config --global user.name "Your Name"
```

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

As a helpful step, you may want to set Git to use your favourite editor

```
$ git config --global core.editor vscode
```

Starting your Journey

First, clone this repository:

```
$ git clone https://github.com/VishweshS/DATA200_Lab12_Fall2023.git
```

Once you have cloned your repository, you should now see a directory called `DATA200_Lab12_Fall2023`. This is your working directory.

```
$ cd DATA200_Lab12_Fall2023
```

```
$ ls
```

For the curious, you should also see the `.git` subdirectory. This is where all your repository's data and history is kept.

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

You will see a couple of git related files, like:

```
branches  config  description  HEAD  hooks  info  objects  refs
```

The Staging Area

Now, let's try adding some files into the project. Let's create one file named `your_first_name.txt`.

```
$ touch vishwesh.txt
```

Let's use a mail analogy.

In Git, you first add content to the `staging area` by using `git add`. This is like putting the stuff you want to send into a cardboard box. You finalize the process and record it into the git index by using `git commit`. This is like sealing the box - it's now ready to send.

Let's add the files to the staging area.

```
$ git add vishwesh.txt
```

Committing

You are now ready to commit. The `-m` flag allows you to enter a message to go with the commit at the same time.

```
$ git commit -m "I am adding one new file"
```

Let's see what just happened

We should now have a new commit. To see all the commits so far, use `git log`

```
$ git log
```

The log should show all commits listed from most recent first to least recent. You would see various information like the name of the author, the date it was commited, a commit SHA number, and the message for the commit.

You should also see your most recent commit, where you added the two new files in the previous section. However git log does not show the files involved in each commit. To view more information about a commit, use `git show`.

```
$ git show
```

You should see something similar to:

```
commit 46bf73350f1ee0a27367f13525ebc2020e2cb8d8 (HEAD -> main)
Author: Vishwesh Srinivasan <vishweshsrini@gmail.com>
Date:   Mon Dec 4 06:49:18 2023 -0500

    I am adding one new file

diff --git a/vishwesh.txt b/vishwesh.txt
new file mode 100644
index 0000000..e69de29
```

A necessary digression

In this section, we are going to add more changes, and try to recover from mistakes. We will need to add some content to the file you just created.

Open the file.

```
$ vi vishwesh.txt
```

Add few lines describing what you liked about this course. Then display the contents of the file to check the contents that you just added.

```
$ cat vishwesh.txt
```

What did we change? A very useful command is `git diff`. This is very useful to see exactly what changes you have done.

```
$ git diff
```

You should see something like the following:

```
diff --git a/vishwesh.txt b/vishwesh.txt
index e69de29..21ea83b 100644
```

```
--- a/vishwesh.txt
+++ b/vishwesh.txt
@@ -0,0 +1 @@
+It was fun teaching the labs!
```

Staging Area Again

Now let's add our modified file to the staging area. Do you remember how ?

```
$ git add vishwesh.txt
```

Next, check the status of the file that you created and modified. Is it in the staging area now?

```
$ git status
```

Undoing

Let's say you want to modify the contents that you just added to the file. One advantage of a staging area is to enable us to back out before we commit - which is a bit harder to back out of. Remembering the mail analogy - it's easier to take mail out of the cardboard box before you seal it than after.

Here's how to back out of the staging area :

```
$ git reset HEAD vishwesh.txt
```

```
Unstaged changes after reset:
```

```
M   vishwesh.txt
```

Compare the `git status` now to the git status from the previous section. How does it differ?

Your staging area should now be empty. What's happened to the changes? It's still there. We are now back to the state just before we added this file to staging area. Going back to the mail analogy, we just took our letter out of the box.

Undoing II

Sometimes we did not like what we have done and we wish to go back to the last recorded state. In this case, we wish to go back to the state just before we added any text to `vishwesh.txt`.

To accomplish this, we use `git checkout`, like so:

```
$ git checkout vishwesh.txt
```

You have now un-done your changes. Your file is now empty.

Branching

Most large code bases have at least two branches - a 'live' branch and a 'development' branch. The live branch is code which is OK to be deployed on to a website, or downloaded by customers. The development branch allows developers to work on features which might not be bug free. Only once everyone is happy with the development branch would it be merged with the live branch.

Creating a branch in Git is easy. The `git branch` command, when used by itself, will list the branches you currently have.

```
$ git branch
```

The `*` should indicate the current branch you are on, which is `main`.

If you wish to start another branch, use `git checkout -b (new-branch-name)` :

```
$ git checkout -b vishwesh
```

Try git branch again to check which branch you are currently on:

```
$ git branch
```

```
  main
* vishwesh
```

The new branch is now created. Now let’s work in that branch. To switch to the new branch:

```
$ git checkout vishwesh
```

`git checkout (branch-name)` is used to switch branches.

Let’s perform some commits now,

```
$ echo 'It was fun teaching the Labs' > vishwesh_feedback.txt
```

```
$ git add vishwesh_feedback.txt
```

```
$ git commit -m " I am adding one new file with feedback"
```

Now, let’s compare them to the main branch. Use `git diff`

```
$ git diff main
```

Basically what the above output says is that `vishwesh_feedback.txt` is present on the `vishwesh` branch, but is absent on the main branch.

Now you see me, now you don’t

Git is good enough to handle your files when you switch between branches. Switch back to the main branch. Try switching back to the main branch (Hint: It’s the same command we used to switch to the `vishwesh` branch above)

```
$ git checkout main
```

Now, where’s our `vishwesh_feedback.txt` file ?

```
$ ls
```

```
LICENSE lab_details vishwesh.txt
```

As you can see the new file you created in the other branch has disappeared. Not to worry, it is safely tucked away, and will re-appear when you switch back to that branch.

Now, switch back to the `vishwesh` branch, and check that the `vishwesh_feedback.txt` is now present.

Merging

We now try out merging. Eventually you will want to merge two branches together after the conclusion of work. `git merge` allows you to do that. Git merging works by first switching the branch you want to into, and then running the command to merge the other branch in.

We now want to merge our `vishwesh` branch into `main`. First, switch to the `main` branch.

```
$ git checkout main
```

Next, we merge the `vishwesh` branch into `main`:

```
$ git merge vishwesh
```

You will something similar to the following output

```
Updating 46bf733..c8aff2f
Fast-forward
 vishwesh_feedback.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 vishwesh_feedback.txt
```

You have to be in the branch you want merge into and then you always specify the branch you want to merge. At this point, you can also try out `gitk` to visualize the changes and how the two branches have merged

To be able to share, we'll need a server to host our git repositiories. GitHub (github.com) is probably the easiest place to begin with.

Create your first GitHub repository

A repository (repo) is a place where you would store your code. The following tutorial: https://docs.github.com/en/get-started/quickstart/create-a-repo will show you how to create a GitHub repo - which you can then share with others

Fork a repo

Go to this tutorial: https://docs.github.com/en/get-started/quickstart/fork-a-repo

To add the URL for the remote repository where your local repository will be pushed, run the following command. Replace REMOTE-URL with the repository's full URL on GitHub.

```
$ git remote add origin REMOTE-URL
```

To share your files with others

```
$ git push origin main
```

To get the latest changes done by others

```
$ git pull origin main
```

References and Further reading:

https://github.com/kuahyeow/git-workshop