CMPT 332 - Operating Systems Concepts
Git Introduction (Part 1)

For the labs and assignments in this course, we recommend using the department's tux machines. To connect to a tux machine, simply use

```
# from a campus computer
$ ssh tuxworld
```

or

```
# from anywhere else
$ ssh nsid@tuxworld.usask.ca
```

Git is a free and open-source version control system designed to handle everything from small to very large projects with speed and efficiency.[1] Git allows you to collaborate with others, keeps your project files safe, and helps you keep your code history organized.

1 First things first: configuring your name and email (if you haven't already)

```
$ git config --global user.name "FIRST_NAME LAST_NAME"
$ git config --global user.email "NSID_OR_EMAIL_ALIAS@usask.ca"
```

2 Git basics

1. Create a local Git repository

   ```
   $ mkdir git-test
   $ cd git-test
   $ git init
   ```

   Git init initializes the current directory as a Git repository. This repository is local, meaning that there is no remote Git server that we use for pushing changes.

2. Create a new file

   ```
   $ echo "Hello World" > hello_git.txt
   ```

[1] https://git-scm.com/

3. Check the status of your repository

```
$ git status
```

Git status shows you the current status of your project with respect to changes.

4. Stage your changes and check the status again

```
$ git add hello_git.txt
$ git status
```

Git add adds changed files to a staging area. The staging area is the area used to bundle changes together and commit them as one atomic unit. If you change your file after staging, you will need to re-add the file to update the one in the staging area.

5. Commit your changes and type a commit message

```
$ git commit
```

Once you have some changes in the staging area that are meaningful to you, you can commit them. *Git commit* allows you to bundle the changes from the staging area into one unit and also include a message that describes your change. Commits should be small, atomic, and meaningful changes to your project. If you find yourself typing a very long commit message to explain the change you made, this is likely because you're putting too many changes into one commit. Git won't complain, but this is not good practice in version control.

6. See the log

```
$ git log
```

Git log shows you a chain of all past commits.

3 Setting up your ssh key with GitLab

Note: ssh is the preferred authentication method for using Git, but if you don't want to use ssh with GitLab, then feel free to skip this step.

1. Find or generate a key-pair.

   If you have previously generated a public/private key-pair, you should find them under **~/.ssh**. Usually, your public key will be named **id_rsa.pub**.

```
$ ls ~/.ssh/id_rsa.pub
```

   If you are unable to find a previously-generated key-pair, then you can create one using **ssh-keygen**.

2. Add your public key to your GitLab account

```
$ cat ~/.ssh/id_rsa.pub
```

   - Copy your public key (make sure not to include any leading or trailing whitespaces), then go to `https://git.cs.usask.ca/` and log in with your NSID.

- From the top right corner, go to **Preferences**, and then to **SSH Keys**.
- Paste your key and add a title, then press **Add Key**.

4 Setting up repositories on GitLab

You can use `https://git.cs.usask.ca/` to create repositories on the department's Git server.

- Use the **New Project** button, then choose **Import Project**.
- Select the **Create blank project** option.
- Set your project name.
- Make sure the visibility level is set to **Private**.
- After the project is created, go to **Project Information**, and then to **Members**
- Under the **Invite member** tab, search for your assignment partner(s) and add them with the role set to **Maintainer** (the highest level of access permissions).

To use your repository, you can clone it using the clone url. Use an ssh link if you are using ssh authentication, or an https link otherwise. The downside to using https links is that will have to enter your credentials every time you interact with the Git server. Cloning the repository is simply done using

```
$ git clone <repo-url>
```

Git clone initializes a local repository, attaches a remote Git server to that repository, and then pulls all commits from the main branch. *Git clone* is equivalent to the following commands

```
$ mkdir <project>
$ cd <project>
$ git init

# add a remote called origin and assign the repo-link to it
$ git remote add origin <repo-link>

# fetch the repository history from "origin"
$ git fetch origin

# checkout branch "master", -b to create if not exisiting, and --track to set
    origin/master as the "upstream branch"
$ git checkout -b master --track origin/master
```

You can now interact with your Git repository as in Section 2. In addition to the commands used in Section 2, you can now also use **git push** to push commits to the Git server, and **git pull** to retrieve any commits done by others.

You may want to either create a repository for every assignment task, or create one repository for all CMPT 332 assignments and use different branches for every assignment task. To create a new branch

```
# create a branch
$ git branch <branch-name>

# switch to that branch
```

```
$ git checkout <branch-name>

# Alternatively, checkout -b <branch-name> will create a new branch and switch to it
    in one step

# upload the branch to the remote server
$ git push -u origin <branch-name>
```

Other assignment partners can now checkout this branch and pull all commits using

```
$ git checkout <branch-name>
$ git pull
```

Collaborating on the same Git branch is possible but not ideal. You can (temporarily) collaborate with your assignment partner(s) on the same branch by making sure to do a **git pull**, then **git commit** followed by **git push** immediately every time you introduce a change to the project. Also, avoid working on the same files at the same time as this may cause conflicts. Branching and conflict resolution is very important and we will discuss this in part 2.

5 Quick reference

- **git add** Tell git to put a file/directory in the staging area.
- **git commit** Commit all staged changes.
- **git status** See the current state of your working copy.
- **git pull** Get the most recent version of the current branch from the git server.
- **git push** Upload your local commits to the remote git server.
- **git log** See the current branch's commit history.
- **man git** Look at the manual for git. There's a lot of other stuff there! You can also open other man pages for specific git commands such as git commit.

Git is an extremely powerful and complex tool. You will never stop learning new tricks and techniques while using git. There are online guides, books, and the man pages to help. `https://git-scm.com/docs` is also a great resource for documentation.