CSE 101
Introduction to Data Structures and Algorithms
GitLab Tutorial

All programming assignments in this class will be submitted through the UCSC GitLab server **git.ucsc.edu**. If you are new to git, spend some time with the introductory material found at ITS GitLab. Follow the login link, then click the register tab and create an account. Your username must be your **cruzid**, which is the string before the **@** in your **@ucsc.edu** email address. Creating your **git.ucsc.edu** account is step zero in this tutorial.

Git is a *Distributed Version Control System*, which briefly put, means that multiple copies of a project are to be maintained in different locations, then integrated and synchronized onto a single server. Git keeps track of all changes to the project on multiple branches of development. At the enterprise level, this might entail the work of hundreds of programmers, all seeking to manage and reconcile their different versions of the project. In this class there is just one programmer, you. Your *project* consists of all of the programming assignments that you submit for grading in this class, and only two copies of the project should be maintained. The *local repository* is the one you directly control and develop. The *remote repository* is stored on the server **git.ucsc.edu**, which is maintained by Information Technology Services (ITS). You will synchronize these two copies by using Unix commands that begin with the word `git`.

If you are completely new to command line interpreters, and to Unix in particular, you have some catching up to do. I suggest you look at lab2 (and possibly lab1) from my CSE 20 from Fall 2020. Just ignore anything having to do with Python.

https://classes.soe.ucsc.edu/cse020/Fall20/lab1.pdf
https://classes.soe.ucsc.edu/cse020/Fall20/lab2.pdf

The remainder of this tutorial will assume that your local repository is to be located in your Unix timeshare account space. You may, if you prefer, maintain the local repo on your personal computing device. The instructions are largely the same as those presented here, except that you would need to first install git on your machine. I recommend that you *not* maintain multiple local repositories, one on the timeshare say, and one on your device, though it is possible. Attempt this only if you are sure you know how.

Set up ssh keys

The next step is to set up a pair of public-private ssh keys to facilitate secure communication between local and remote repositories. Log on to the Unix timeshare **unix.ucsc.edu** (see the above CSE 20 links if you don't know how to do this), and type the following command.

```
$ ssh-keygen -t rsa -b 2048 -C "label"
```

The dollar sign `$` represents the Unix command prompt, and you do not type it. Also `"label"` is a comment that you choose. This comment can be anything you like, but typically it is used to identify the computing device you are on. In this case `"timeshare"` would be an appropriate value. If you are setting up a repository on your own computer, you might choose the name of your machine as the label. The result of the above command will be

```
Generating public/private rsa key pair.
Enter file in which to save the key (/afs/cats.ucsc.edu/users/a/cruzid/.ssh/id_rsa):
```

or something similar (`cruzid` will be your CruzID and the letter `a` may be something else). Press return to select the default location `cruzid/.ssh/id_rsa`, (the file `id_rsa`, within the directory `.ssh`, within your home directory `cruzid`.) Navigate to the directory `.ssh` (do `cd` then `cd .ssh`) and type

```
$ cat id_rsa.pub
```

to see the public half of the pair. The private half is in the file `id_rsa`. Do not share the private key with any person or computer, and do not alter either file.

Now login to **git.ucsc.edu** and open the web portal. In the upper right corner of the page is a pull-down menu containing information related to your account. Open it and select **Preferences**. Go to the left-hand pane on the settings page and select **SSH Keys**. Copy-paste the public key in `id_rsa.pub` into the `key` box in the web portal. Give it a `title` (and an optional expiration date if you like), then press `add key`. You have now established a secure method of communication between your account on the timeshare and your account on **git.ucsc.edu**.

Set up your local repository
Return to your home directory on the timeshare (by typing `cd`), then type

```
$ git config --global user.name "first last"
$ git config --global user.email "cruzid@ucsc.edu"
```

where `first` is your first name, `last` is your last name and `cruzid` is your CruzID. Now create a subdirectory within your home directory called `cse101`, where you will keep all of your work for this class, then cd into it.

```
$ cd            # return to your home directory
$ mkdir cse101  # make a subdirectory called cse101
$ cd cse101     # change your working directory to cse101
```

Note that everything after `#` on a line is a comment and need not be typed. Now do

```
$ git clone git@git.ucsc.edu:cse101/fall23/cruzid.git
```

where as usual, `cruzid` is your CruzID. If you get an error message resembling

```
Cloning into 'cruzid'...
remote:
remote: ========================================================================
remote:
remote: The project you were looking for could not be found.
remote:
remote: ========================================================================
remote:
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

it means that I have not yet created your repository on **git.ucsc.edu**. I will be running a script periodically (once a day to start with) that creates student repositories, but I can't create a repository for you if you don't

have an account. This is why I run it once a day. If you have this error, you should pause and try again the next day.

Assuming you did not get an error, the above command should produce the following output.

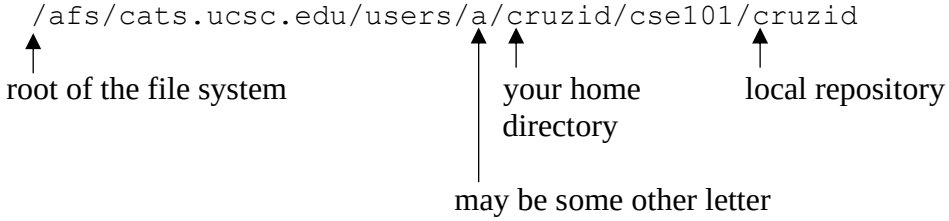
```
Cloning into 'cruzid'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 10 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (10/10), done.
```

Once this is done, a new subdirectory within `cse101` called `cruzid` has been created.

```
$ ls     # list the contents of the current directory
cruzid
```

This subdirectory is your local repository. Make `cruzid` your current directory, and just in case you have become confused as to your location, type `pwd` to see the full path name of the repository.

```
$ cd cruzid  # change directory to cruzid
$ pwd        # print working directory
/afs/cats.ucsc.edu/users/a/cruzid/cse101/cruzid
```



Currently, this directory contains several files placed there when your repository was created. To see the full contents do

```
$ ls -a  # same as ls, but include files that begin with dot .
```

You can ignore the contents of this directory at the moment.

Git maintains multiple *branches* of your repository. Branches allow developers to make changes in a self-contained environment, isolated from the rest of the repository. Think of different branches as being separate copies of the whole repo. Changes made on one branch do not affect the other branches. When your repo is created, it has exactly one branch, called `main`. You will create a separate branch for each programming assignment in this course. Our automated grading systems will use these branches to apply separate testing environments for each assignment.

The command for changing to a new branch is `git checkout`. The command

```
$ git checkout -b pa1
```

Creates a new branch called `pa1` and switches you from `main` to `pa1`. Any alterations you now make to the repo will affect only the `pa1` branch. To switch back to `main`, one would do `git checkout main`, but don't do this now.

Create a subdirectory for programming assignment 1, called `pa1` and `cd` into it.

```
$ mkdir pa1
$ cd pa1
```

The new directory `pa1` is within the directory `cruzid`, but is not yet a part of the repository. However, we cannot add an empty directory to a repository, so we must first insert some files. Eventually you will place all of your files for `pa1` here, but for now a couple of empty files will suffice.

```
$ touch file1
$ touch file2
```

The Unix command `touch blah` will create a new empty file called `blah`, if `blah` does not exist. View the manual pages for `touch` (do `man touch`) to see what it does if file `blah` already exists. Now we have a directory within the pa1 directory containing two empty files. To add it to the repository, do

```
$ git add .
$ git commit -m "initial commit on pa1"
```

The dot `.` in this context means your current working directory, which you'll recall is `pa1`. The command `git add .` places the directory `pa1`, with its contents `file1` and `file2`, into a staging area called the *index*. The command `git commit` changes the local repository so that it now includes the new items. The option `-m "message"` attaches a comment to this commit, so users will be able to follow the history of changes. At this point, all that has been changed is the local repository. To synchronize with the remote repository, do

```
$ git push -u origin pa1
```

This command performs the initial synchronization between your local repo and the remote repo on **git.ucsc.edu**. Subsequent synchronizations can be done by doing just `git push`.

Note that there are two things here called `pa1`. One is a *directory* in your account space on the timeshare, constituting part of your local repository. The other is a *branch* of your local repo. All of this is synchronized with the remote repo by the `push` command. In general, it is not necessary that directory names match branch names, but you must maintain this convention to insure our grading scripts do not get confused.

Observe that `add` has no terminal output, but `commit` and `push` do, which was not included above. To verify that the remote repository has changed, go the web portal on **git.ucsc.edu** again, go to **projects** in the upper left corner, then **your projects**. You can see the full contents of **cse101/fall23/cruzid** by navigation in your browser.

This combination of git commands: `add`, `commit` and `push` will be your main tools for doing subsequent submissions to `pa1`. For instance, to add a third file, do

```
$ touch file3
$ git add file3      # git add . would be fine here too
$ git commit -m "add file3"
$ git push
```

and observe the changes to the web portal. Now try deleting a file using `git rm`, then `commit` and `push`, as before. Intersperse the command `git status` between the others to see a readout of changes to the repository.

```
$ git rm file1   # remove file1
$ git status
$ git commit -m "delete file1"
$ git status
$ git push
$ git status
```

Let's add some content to one of the remaining files, and use `git diff` to display the difference. A simple way to append text to a file is `echo` with output redirect >.

```
$ echo "some content for file2" > file2
$ cat file2
some content for file2
$ git diff
diff --git a/pa1/file2 b/pa1/file2
index e69de29..877005d 100644
--- a/pa1/file2
+++ b/pa1/file2
@@ -0,0 +1 @@
+some content for file2
```

The output of `git diff` is a little bit cryptic, but this article may help explain it somewhat. Ordinarily we won't run `git diff` often, but it's good to remember that all git does, at the most base level, is to track changes to files and directories. The same combination of `add`, `commit` and `push` will synchronize the local repository with the remote.

```
$ git add .
$ git commit -m "content for file2"
$ git push
```

At this point, you can add your source files for `pa1` (and also remove `file1` and `file2`), then perform the now familiar `add`, `commit`, `push` combination.

When it is time to start working on pa2, you will create a new branch with

```
$ git checkout -b pa2
```

then

```
$ mkdir pa2
$ cd pa2
```

Then use the `add`, `commit` and `push` commands to place new work in the local repo, and synchronize with the remote. Repeat the whole process for `pa3`, … , `pa8`.

Attend the TA/tutor/instructor office hours to get help with this tutorial if anything is unclear. Git has a rich command set giving a fine level of control over your projects, which we've only glimpsed. The Git Book is one place to continue learning. Another resource on Unix is the git tutorial.

```
$ man -7 gittutorial
```

There are many others.