UDACITY COURSE: 1 HOW TO USE GIT AND GITHUB

https://classroom.udacity.com/me

PREREQUISITES:

Command line: something that helps us to interact with the computer to perform actions such as listing and removing files for example, in a language that computer understands. Babun is a free command line program, easy to install and run the commands.

ls –l ~/Desktop : ls -> utility, action needs to be done | -l -> conditions for utility | ~/Desktop -> argument, gives an idea of how to do the action mentioned.

Example:

man $UTIL – gives information on UTIL, UTIL can be any utility such as ls, cd or even man (man ls) .Use up and down arrow to scroll through the document and q to go back to the command line

Never use: sudo rm –rf – continually removes multiple files from the location. :(){ :|:& };: - keeps on running thus taking up the entire computer resources.

Cygwin is a UNIX like environment and command line interface for windows. Pact is package manager used to install, upgrade, and search the Cygwin package. This is provided in babun. Babun also supports mintty which is in xterm format, which is a standard emulator(hardware or a program which can act as another hardware or a program) for windows. There are two kinds of shell (interactive interface between user and operating system) that is provided in Babun, bash and zsh (default), which provides UNIX, software development tools and much more.

Bash shell is invoked when you invoke minty, otherwise zsh shell is invoked. To find out which shell is currently under use, we can use the command echo $SHELL. Bash doesn’t emulate the korn shell like zsh does.

Babun also provides a proxy server which acts as a gateway between local networks and a bit larger networks called as the internet. Also it provides a number of programming language support such as for python and Perl, UNIX programming tools etc.

Microkernel is the minimum amount of software that can implement the mechanism to run an operating system. Plugin architecture in babun is something that helps in adding additional features to the program, thus helping in customization.

Owning a public github is an advantage for the developers. It helps in keeping track of multiple files, track bugs by reverting to previous working versions of the file, collaborate with other developers of the project, especially in professional environment.

Version Control:

Makes it easy to save and/or compare different versions of the same file at different points of time. It includes features like restoring to the previous version of the code. The files are saved at a common repository, which means that we don’t have to send huge chunks of code in .zip files.

GIT: - version control platform GITHUB: code sharing and collaboration platform.

To find differences between two files, we use the command FC, which stands for file compare in command line prompt.

Q: is git and babun different command line prompts, that is the difference between the two ?

.bashrc is a shell script that runs automatically when it is invoked interactively when git bash is opened.

echo 'alias subl="C:/Program\ Files/Sublime\ Text\ 2/sublime\_text.exe"' >> ~/.bashrc

The command here sets ‘subl’ to the sublime\_text.exe file located in the path starting with “c:” and appends (>>, > does overwriting)it to the shell script .bashrc located in home directory when the git bash is opened.

Git uses any editor in it and can be used offline. Git needs manual saving, allowing user to create the checkpoints so that each individual save represent logical change. These user created checkpoints are called commits. When a commit is created, user is expected to provide a commit message. This message can prove meaningful if the user provides a logical message. Each commit in a git has an ID, author, date, and a message associated with it.

**Git bash" is a msys shell included in "Git for Windows", and is a slimmed-down version of Cygwin (an old version at that), whose only purpose is to provide enough of a POSIX layer to run a bash.**

* **msysGit** - is the name of this project, a build environment for Git for Windows, which releases the official binaries

To find the difference between two git commits, use the command git diff along with the commit ids of the versions of file that you want to see the difference for.

Git diff old\_id new\_id

REFER TO THE MAP OF VERSION CONTROL

Make one commit per logical change. If the two code changes are associated with a particular functionality, make one commit. If the two code changes are at different parts, make two different commits. Git allows us to include a message indicating what change is performed.

Repository is a collection of files that needs to be tracked together. When you make change to one file in a repository and choose to commit the change, the version of all file with or without the changes gets committed or saved for that particular time.

Git log –stat -> gives the list of files that have been changed in each commit.

Git –version -> version of the git being used.

In order to copy the entire history of files associated with a project, we use the command git clone. Downloading would just get you the recent commit codes. The command is:

Git clone url

Git config –global color.ui.auto – shows the changes in the file using colors.

Git errors and warning:

Should not be doing an octopus: octopus is a strategy that git uses to combine different versions of code together and this error is issued when you try to use this strategy in an inappropriate situation.

You are in a detached head state: Head indicates the commit in which you are currently in. Detached head state warns that you are going to the previous version of commit.

Panic!(The impossible has happened) : not a warning message used by Git.

Git checkout: restoring to a previous version of the code. This helps in scenarios where you have a bug introduced and not sure of the version where the bug is seen. On giving this command, all the files will revert to the stage when the commit was made.

Git checkout id -> check out the file.

If you have checked out an older commit like an example the commit with “revert control” message and if we type the command git log/ git log --stat, this shows the commit with “revert control” as the recent commit. Now we know that this was not the recent commit, so to return back to the recent commit (which in this example was adding color, which can be noted before checkout by executing the command git log stat) we can do git checkout id.

\*in the command prompt denotes a file with changes that is not committed yet.

In order to complete the commit, we have to enter a commit message which normally is included in a text editor. The editor is sublime that we will be using.

The following are the things present in .bash\_profile:

git-completion.bash – used for the tab completion

git prompt.sh – used for the git related stuffs such as commit id.

And the .bash\_profile includes stuffs such as inclusion of \* when you change a file, color of the prompt and so on.

Git config - -global core.editor “ ‘D:\soumya/sublime\_text.exe’ –n – w” (-n = new, -w = git allows you to close sublime before continuing)

Anything with a . filename is hidden in the folder and wont show up most of the case.

Git config - global push.default upstream

Git confi - -global merge.conflictstyle diff3

If student A specifies the location of a directory or repository, student B can copy using the command scp.

Copying can be done on any directory, while git clone works only on the git repository.

0 commits are created when you create a new git repository, this is because there is only manual commits in git.

**Checking out an earlier commit will change the state of at least one file.**

This is sometimes true. Git doesn't allow you to save a new commit if no files have been updated, so you might think this is always true. However, it's possible to do the following:

* Save a commit (call this commit 1).
* Update some files and save another commit (call this commit 2).
* Change all the files back to their state during commit 1, then save again (call this commit 3).

This sometimes happens if commit 2 contained a bug, and it's important to fix the bug quickly. The easiest thing to do might be to remove all the changes introduced by commit 2 to fix the bug, then figure out how to safely reintroduce the changes later.

At this point, commit 3 is the latest commit, so if you checkout commit 1, none of the files will be changed between commit 1 and 3.

Git repository is very similar to a directory, but it also has metadata (data about the data) about the history of repository when it was created (.git directory). The metadata gets transferred when we clone the git repository.

In order to create a new git repository, you can create a new directory for that or you can move into an already existing directory and initialize it. Gt doesn’t create a commit for you, you have to create it by yourself as you might not want to commit everything in the directory and you might not be able to provide the git commit message if git did the first commit for yourself.

Git init – initializes an empty repository (this also creates the .git folder)

Git repository is specific to the directory.

Git status: command that shows which files have changed recently after the last commit. For a new repository, it shows that you are ON A MASTER BRANCH. Git shows the most recent commit if you have one, else it shows INITIAL COMMIT. Any files in the new repository will be under the untracked file category as git is not tracking their history yet.

Once you have a working directory with files in it, we have to add/commit the files into the repository (which is created in the directory.) In order to do this, we have to pass the files through an intermediate area called the staging area. You can add one logical file(files with some similarity) into the staging area. Then commit the files from staging area to the repository.

Your local repository consists of three trees maintained by git. The first one is the working directory which holds the actual files. Second is the index which act as the staging area and third is the head that points to the last commit that you have made.

Git add filename – adds the file name to staging area. (Can add multiple files at the same time.)

Run git status, which shows that the files added to the staging area will be shown under the category of changes to be committed.

To remove a file from the staging area, you can use the command

Git reset filename – removes the file from staging area and would be present in the working directory. Working directory has an empty repository initialized in them but do not contain any tracked files in them. This can also be done using the git rm –cached filename command.

Git commit –m “commit message” – lets you add the commit message, instead of git commit opening up an editor. The message should contain:

Type: subject

Body

Footer

Write the message as a command, rather than as a statement.

Once the commit is made, git log shows the output where your message is displayed. And git status shows any untracked list of files.

<https://github.com/github/training-kit/blob/master/downloads/github-git-cheat-sheet.pdf>

For the asteroids directory, you have an error, for which you find the commit id which introduced the bug. Now you take a diff of the commit id that caused the bug and the previous one, which shows the deleted line which is the reason for the bug. You copy paste the line in the file(game.js here), and save it. Now

git status ---- shows the file that you have modified which are not staged yet.

Sarath ((3884eab...) \*) asteroids $ git status

HEAD detached at 3884eab

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: game.js

no changes added to commit (use "git add" and/or "git commit -a")

Sarath ((3884eab...) \*) asteroids $

Now, modify another file for example index.html for differentiation, which we won’t be committing yet.

If you have made some changes in the files which are in the working directory and not staged yet, you can find out the difference or what modification that you have made in the file by giving the command GIT DIFF WITHOUT ANY ARGUMENT.

Suppose you have made changes to two files in the working directory. Now if you do a git diff (with no arguements) you would get the modification that you have made for both the files. Adding one file into the staging area (using the command git add) and now doing a git diff will show the modification done on one file which was not added in the staging area.

Now once you have added the files that has the changes into the staging area, you might want to compare that file with its copy in the repository. That difference can be shown by git diff –staged. If you have some changes in the file that you need to discard that either belongs to working directory or staging area, you can use the command git reset –hard. (we cant never get the changes back once we run this command)

Git checkout master is the command used to go back to your recent commit without knowing the commit id.

Branches create a new variation of the linear work history in which we can try experimenting. Master is the name given to the main branch in most git repositories. When you create a repository, git creates a master branch for you. We can label the branches that are not master. Detached head state tell you that you are in a commit that has no name. Just like a commit, you can check out a branch which has a readable name.

When you checkout a branch and make a commit in it, the branch label automatically updates to new commit. This new updated branch stays checked out. This is how we stay on the master branch without knowing anything about the branches in the later on phase. Current last commit is the tip of the branch. We can have multiple branch labels associated with a single commit, but a new commit in branch will be done to the one that is checked out. The process of combining two commits (can be in different branch) is MERGING.

Git branch – shows the current branch

Git branch branchname – creates a branch with name branchname.

Sarath (master) asteroids $ git branch

\* master

(shows that master is the current branch that is checked out)

Remote branches are those that are created by others. On check out, these remote branches behave like any other normal branch.

To visualize the branch structure : git log – graph –oneline brachname

Git log doesn’t show you every commit made, each commit knows its parent. It stores the reference of the commit that was checked out when it was made. Commit doesn’t care about branch names. A branch can trace back from Its most recent commit to the parent commit which is the initial commit. They would not have access to the branches in between.

To get the detached head message we checked out a commit. Head means that we are in the current commit.

Git checkout –b new\_branchname – (git branch branchname+ git checkout branchname)

Git has the record of the code before the two code diverged. Parent of the merged commit from master and ‘x’ branch will be both files. After merge master branch gets updated with the ‘x’ branch info. (merging x into master). Once the merge is completed, the master branch also includes changes from coin branch. We can also delete the ‘x’ branch(removing the label) after the merge and all the code in x can be accessed through master. All the commits will be shown in terms of their time stamps.

If a branch is deleted and leaves some commits unreachable from existing branches, those commits will continue to be accessible by commit id, until Git’s garbage collection runs. This will happen automatically from time to time, unless you actively turn it off. You can also run this process manually with git gc.

Git merge master coins --🡪 merged version of the code.

To see the difference between a commit and its parent, we can use the command : git show commit id

To delete a branch : git branch –d coins (only delete the label, not the commits)

Git will notify if there is any change in a common area while merging than solving it by themselves.

Once you see a conflict, make the changes to the file and save it. Then do git status, git add, git commit. And then merge it.

Now the output git log –n 1 – gives you the file which underwent the conflict.

Github in general supports many formats of data projects in it. We can contribute to projects if the people maintaining the projects allows you to do so. Public repositories are free and private will be charged. We can send changes to github via command line and also directly access them online.

Github: username – cognition143, mail id : [soumyasarathons@gmail.com](mailto:soumyasarathons@gmail.com), password : unnipomi143!

Github is built on git.

On interacting with github, the information sent and received is in terms of git commits, so changes must be committed from local working directories before sending them. The working directory and staging area of a github is not directly accessible as they are hosted on the github servers. We need to choose WHEN AND HOW to get the two versions synced. This is because if the github syncs and publishes every single commits, it might make no sense as a new feature can include multiple commits.

To get your commits on github, cloning doesn’t work. We have to create new repository. There is a concept of remote repository in our working directory or computer that stores the location of github repository where you have to send and receive files from. You can push and pull data with the remote directory.

An easy way to send data over would be push the master branch rather than pushing in a number of commits. Git wouldn’t send a data that is already in the github.

Go to the website, check for your username(top right, and click on +) – next to owner give your repository name and leave it as public – do not select README if your repository will have data, if your repository is empty, you cannot clone it.

Now you have to have a remote repository in computer to store the location of github rep.

Git remote (to view and create remote), you can use ‘origin’ as your remote repo:

Git remote add origin \*url-for-remote\* (git remote add origin https://github.com/cognition143/reflections.git)

Sarath (master \*) reflection $ git remote

origin

Sarath (master \*) reflection $ git remote -v

origin https://github.com/cognition143/reflections.git (fetch)

origin https://github.com/cognition143/reflections.git (push)

Now to push data we use the command: git push origin master

Now your git repository will be updated with the changes you have introduced. Git log in local repository = click on the commits.

It also create and edit plaintext files directly on the github website, without making the same changes to computer repository. We can also modify the github from another computer (like by some other user).

We can create a new file in the github repository also. Now to update the local repository with the

Github repository, we pull the changes into the file which needs to be updated (this file should be checked out)

Git pull origin master (local and github repository name can be same or different)

To modify an existing piece of information in the github to our taste and to keep the original version intact in the github, we use forking concept. This does a cloning of a repository to your own github repository rather than downloading the code to your local machine, you editing, committing and pushing those into the repository. If the changes are simple enough, we can edit on the github directly. Github keeps a track of number of time forking has been done on your repository, all the forks link back to the original. It makes it easier to suggest changes back to the original repository.

Go to the page you want the forking, click on fork button, a copy of the repository will be available on your repository. Now clone the repository to your local machine and make the changes you want. We do not have to create a remote repository to point to the github url, because the git would have automatically created a remote to the repository where we cloned from.

Sarath version-control $ cd recipes/

Sarath (master) recipes $ ls -a

./ ../ .git/ cake-recipe.txt chili-recipe.txt frosting-recipe.txt README.md

giSarath (master) recipes $ git remote -v

origin https://github.com/cognition143/recipes (fetch)

origin https://github.com/cognition143/recipes (push)

Sarath (master) recipes $

If you want somebody along with you to collaborate for the work, you would give them the push access. To add someone, go to setting, collaborators, add their github name.

After you ran git push, your change should have existed locally and on your fork.

When you have an extra branch added at both the git repo and your local machine (change in both place), pulling a change from git repo can cause the change in local to get cancelled. So we have a conflict there. We will have to merge the local and remote version of the branch.

Now I have made some changes in recipe(adding corriander) while at the same time sara has deleted the cumin from the chilli recipe. I will do :

Sarath Downloads $ bash sarah\_changes.sh https://github.com/cognition143/recipes

Cloning into 'recipes'...

remote: Counting objects: 19, done.

remote: Compressing objects: 100% (3/3), done.

remote: Total 19 (delta 0), reused 2 (delta 0), pack-reused 16

Unpacking objects: 100% (19/19), done.

warning: LF will be replaced by CRLF in chili-recipe.txt.

The file will have its original line endings in your working directory.

[master 94374cf] Remove cumin from chili

1 file changed, 1 insertion(+), 2 deletions(-)

Counting objects: 3, done.

Delta compression using up to 4 threads.

Compressing objects: 100% (3/3), done.

Writing objects: 100% (3/3), 309 bytes | 0 bytes/s, done.

Total 3 (delta 2), reused 0 (delta 0)

remote: Resolving deltas: 100% (2/2), completed with 2 local objects.

To https://github.com/cognition143/recipes

75df430..94374cf master -> master

This step added sara’s change to the fork. While in the local repo I have only my changes.

When you have a remote set up, git stores all local copies of remote branches. This will contain the state of each branch as you pushed or pulled them.

Git stores local copies of all remote branches, containing their state at the time of pushing or pulling them. When you clone a repository the git automatically creates a remote, known as origin and creates a local branch called as master that points to the same branch as in remote. Local copy of last known position of that branch in remote (local copy of remote branch),which will have the name as includes the name of remote / name of the branch .(eg:origin/master). When you make changes in the local and commit it, the pointing towards the branch(origin/master and the master in remote) isn’t moved until it is communicated via push to both local and github while the master gets updates to the new commit. Similarly if we make change in the branch in github, unless we pull it, the updation with new commit wont happen.

If we have changes in both the local and remote, what we can do is we can get the changes from the remote through fetch command, which will update the local copy of remote branch, leaving your actual copy which is the master. Now once you have the changes from the repository to your local copy of remote branch, we can use git merge to combine master and origin/master.

Git pull -> git fetch +git merge

Git fetch origin - will update all the local copies of all branch for the origin remote.

Git log origin/master or git diff origin/master – gives you the status

In order to merge the new change in repository (which is present in your origin/master branch after the fetch) and the change you have introduced, do:

Git merge master origin/master

Fast forward merge : involves merging two commits where one is the ancestor of the other i.e. one is reachable by the other.

If you are working on a different branch (different-oil), while git push you need to add the different branch name(git push origin different-oil)

To create pull request, you have to go to your changes, click on your recipe than larry’s to which you like to merge and click create new pull request. .

If someone else makes a change in your pull request, git will mark it as a merge conflict and one of the programmers will have to resolve it. So you can go ahead and merge one of the changes (say eg merge1) and you can safely delete the branch merge1. Now we can remove the merge conflict my downloading the code into our system. MERGING WITH THE BUTTON ON GITHUB WILLALWAYS MAKE A COMMIT ANYWAY EVEN IF NO EXTRA INFORMATION IS GIVEN BY THE MERGE COMMIT. (usually if the branch is getting merged with the master, the tip gets updated with the master) while making changes or resolving merge conflicts within a pull request, do not go and merge the master branch (which contains the changes of the other person) with your changes. On doing so you will do the merging without the review of the other person who made the changes along with you in the pull request. So you have to make changes into your new branch rather than the master and push the new branch into the repository. This activity will update the pull request. So the other person who made the changes into the file can review the change and merge it with the master.

In order to redirect from one page to another with links as we did in the final project use the github links to go between the markdown pages.

Rm –rf .git 🡪 would remove the entire git repository from directory.

How to use multiple git accounts from the same machine, this is for ssh and this generates a new key for the new account that you set up in github:

<https://code.tutsplus.com/tutorials/quick-tip-how-to-work-with-github-and-multiple-accounts--net-22574>

Now if you are using https: you have to set a global username and email address for this particular repository , else it would take the first username and email address

$ git config user.email "*email@example.com*"

$ git config user.email

git config user.name USERNAME

git config user.email USERNAME@example.com

git remote set-url origin <https://USERNAME@github.com/USERNAME/PROJECTNAME.git>

also, just in case, you can check your credential manager to see if any github account with its username and password is saved, juts remove it and then try using the multiple accounts. It just worked for me ☺