**Git**

Getting started:

1. Go to github.com and make an account/ log in into an existing account
2. Click on “Set up Git” from the GitHub Bootcamp menu, download and install the application
3. Open the GitHub program on your machine and log in with the same account
4. On the next step enter a display name and a contact e-mail for other people to use if they want to contact you regarding your code.
5. Go to Start - > Git Shell, open it and type:

*git config --global –edit*

Notepad should open with the following content:

|  |
| --- |
| [user]  name = “The name you entered in step 5”  email = “The e-mail you entered in step 5” |

IF THAT IS NOT THE CASE PLEASE WRITE IN THE SHELL:

1. *git config --global user.name "YOUR NAME"*
2. *git config --global user.email "YOUR EMAIL ADDRESS"*
3. Check again with: *git config --global –edit*
4. Now we have to make sure you have the SSH keys properly set. Run the following commands in the Git Shell:
5. *ssh-keygen -t rsa -C “your\_email@example.com”* – this will generate a new SSH key for you. **When it asks you where to save it just press ENTER.** Then enter your passphrase.
6. *ssh-agent –s* - this starts the SSH agent in the background
7. *ssh-add ~/.ssh/id\_rsa –* This will add the newly generated SSH key to the agent.
8. *clip < ~/.ssh/id\_rsa.pub –* This will copy the content of the “id\_rsa.pub” file to the clipboard.
9. Open Settings on the github.com
10. Click on “SSH Keys” on the left
11. Click on “Add SSH Key”, then type any title
12. Paste the key and click “Add key”

Test that everything is working:

1. Open Git Shell and type:

*ssh -T* [*git@github.com*](mailto:git@github.com)

1. Type “yes” after the warning
2. If it then says then everything is ready.

*Hi “username”! You've successfully authenticated, but GitHub does not*

*# provide shell access.*

**We will then add you to the repository of your team.**

Once you have been added to the repository of your team:

1. We will give you a link
2. Go to the GitHub application, click on “+” on the left and choose “clone”
3. Click on the “gutechsoc-hackathon”, find your teams name and clone that repository.

**BASIC COMMAND FLOWS**

The idea behind GitHub is that it manages changes to your code so that you can always go back to any previous set up that worked. This way you can experiment, add, remove, change code and feel save that if you mess up you can always go back to the previous “working” state. Git is also perfect for big projects since it allows code changes to be propagated to any computer quickly and efficiently.

This is achieved with a few commands and concepts which you have to understand in order to start using Git:

* Repos – this is basically your files with code. There are at least 2 repos – 1 local (on your machine) and 1 remote (on the github server). The idea here is that you do changes on your local repo and then you “commit” (send) them to the remote repo. Changes on the remote repo can be then downloaded by the other members easily. A repo consists of 1+ branches.
* Branch – A branch in Git is simply a lightweight movable pointer to one of the commits. The default branch name in Git is “master”. As you initially make commits, you’re given a master branch that points to the last commit you made. Every time you commit, it moves forward automatically. A repo can have lots of branches as part of its structure.
* Commit – this is the action of moving changes in your local repo to the one on the github server. There is a certain procedure you have to go through to make a commit. Each commit is connected only to the previous commit.

The GitHub application manages the commit process for you. When you think you have made all of the necessary changes and want to commit, just go to the GitHub application and press Sync. This will update your current local repository by downloading any “committed” change by your teammates and then (if there are no conflicts) it will upload your changes.

**CONFILICT MANAGEMENT** – A conflict is when git algorithm does not know which change to save. This usually happens when 2+ people change the same lines of code at the same time. If conflicts arise when committing the app will tell you where exactly are the conflicting lines of code. You will have to go to this file manually and edit it. A conflict-marked area begins with <<<<<<< and ends with >>>>>>>. The two conflicting blocks themselves are divided by a =======. It will look something like this:

<<<<<<<< HEAD

Original code

===============

Modified code

>>>>>>>>>>>

To resolve the conflict decide which code you want to keep and delete everything else so that only the desired block of code remains.

**Good practices:**

Have at least 2 branches on the github server.

1. **PRODUCTION** – this is where you will keep versions of code which are tested and working.
2. **DEVELOPMENT** – this is where you will keep versions of code under development.

The idea is that you develop on your local repository, then do a commit to the DEVELOPMENT branch and so do all of your teammates. Then when you have completed a functionality, tested it and it works as you want it to, you “save” this progress by shifting this code to the PRODUCTION branch – this is called branch merging. Then you begin working on a new functionality on the DEVELOPMENT repo and so on.

This is a good practice because it separates your code into fully working code and under development code. This is also a precaution for any big merge conflicts and also in the situations where you have had 2-3 good commits and then something crashes.

**USING GIT FROM THE COMMAND LINE ( use Git shell)**

(Try to understand this as it will give you a better understanding of the commit process)

**GIT WORKS ON THE FILES DIRECTLY THEREFORE FILE-BASED COMMANDS (like git pull) WILL CHANGE THE FILES ON YOUR SYSTEM. THIS CHANGE WILL BE VISIBLE IN YOUR DEV ENVIROMENT AFTER YOU REFRESH IT.**

Creation:

* git clone “linkToRepo” – clone(download) existing repo
* git init – create a new local repo in the current dir

Local changes:

* git status – gives a list of changed files in comparison to the remote repo
* git diff – displays the changes themselves in the command line
* git add <pathToFile> - add this file to a commit
* git commit – push changes to tracked and added (with “git add”) changes to remote repo

Remote repo:

* git branch – displays a list of all existing branches
* git checkout <branch> - switches your current HEAD branch to the one in <>
* git branch –d <branch> - delete branch
* git pull <branch> - download changes from the branch and merge them with your local repo
* git push <branch> - push local changes on the remote repo
* git merge <branch> - merge <branch> into your current HEAD (your current branch)

Undo:

* git reset –hard HEAD – discard all local changes in your working directory (local rep)
* git checkout HEAD <file> - discard local changes in a specific file
* git revert <commit> - revert a commit (by producing a new commit with contrary changes)
* git reset –hard <commit> - reset your HEAD pointer to a previous commit and discard all changes since <commit>
* git reset <commit> - same as above but preserves all changes since <commit>
* git reset –keep <commit> - same as above but preserves uncommitted local changes

**Commit process.** Execute the commands in the following order:

1. git pull <branch> - to download changes from the remote repo and merge them into your local repo
   1. If there are conflicts resolve them now
2. git status – to see which files are tracked and not-tracked and in which files there are changes
3. git diff – to see exactly what the changes are.
4. git add - to add the files you would like to push to the remote repo
   1. git add <filePath> - to add a specific file
   2. git add –a – to add all TRACKED files
5. git commit –m “Good-Explanation-of-the-changes” – to create a commit
6. git push <branch> - to upload your changes to the files you specified on step 4

If you have any problems or questions please contacts us and we will try to help you as much as we can.

Thank you for coming. We hope you have a great time.