1. How to Download and Install Git

I'm going to be teaching you guys how to download and install Git and also how to use it from beginner to expert.

Git (G) is pretty much a program that lets you keep track on changes in files (in a project). It lets you manage all of these versions of the same exact file and it does it a lot better than you do (probably).
If you ever want to go back to the original one, it lets you do that incredibly easy and you don't have to keep renaming stuff.

Intricacies - zawiłości

It's one of the best programs for managing source code, doing and undoing fixes etc.
You can actually use this with any file, Word docs, Photoshop files, text files, Microsoft docs. It's just programmers use it most often, but anyone can use it.

Downloading G. Bucky has version 1.9.2. I'm downloading 2.15.1.
Select 'Git from Git Bash only', cool option.

Git Bash is like a command line for Windows except this is actually UNIX style.

```
$ git --version
git version 2.15.1.windows.2
```

The command whenever you type it, it kind of sounds like what you want to do, like 'get the version of my git'.

2. Config Our Username and Email

Before creating ourfirst git project we need to fill out our username and email. Basically it's a way of creating an account. The reason we need to do this is because whenever we're working on a git project with a bunch of people, we're all going to be making changes to the core project. For example, if you're all building a website and you're going to have a programmer, a web developer, a graphic artist all making changes to the same project. So it's easier to sort things out and see who did what and that's why everyone needs their own account.

```
$ git config --global user.name "Miki"

$ git config --global user.email "mikolaj.myszka@gmail.com"

$ git config --list
```

```
$ clear
$ git help
$ git help commit
```

3. Creating Our First Repository

```
$ cd ~
```

```
$ cd .. wazny odstep przed kropkami

$ pwd

$ ls

$ cd Tuna

$ git init
```

4. Commit

```
$ ls -la
total 84
drwxr-xr-x 1 Miko³aj 197121 0 Dec 23 17:30 ./
drwxr-xr-x 1 Miko³aj 197121 0 Dec 23 17:34 ../
drwxr-xr-x 1 Miko³aj 197121 0 Dec 23 17:30 .git/


$ git add .

$ git commit -m "This is the first commit"
[master (root-commit) 4fc780b] This is the first commit
 1 file changed, 1 insertion(+)
 create mode 100644 first.txt
```

5. Adding Files and the Commit Log

```
$ git log
commit 4fc780b99af5ebff9c7d354d43802d55e0a5917f (HEAD -> master)
Author: Miki <mikolaj.myszka@gmail.com>
Date:   Sat Dec 23 17:59:02 2017 +0100

    This is the first commit


$ git log --author="Miki"
commit 4fc780b99af5ebff9c7d354d43802d55e0a5917f (HEAD -> master)
Author: Miki <mikolaj.myszka@gmail.com>
Date:   Sat Dec 23 17:59:02 2017 +0100

    This is the first commit


$ git log --author="Wendy" --> there is no Wendy user

$ git status
On branch master
nothing to commit, working tree clean
```

It says that there are no files that you're working on that git isn't keeping track of. In other words, if you created a file, but you didn't commit it yet, then it would give you an entirely different status.It pretty much compares your repository (which is that main project), against your working directory which is just your local copies that it's working on and it tells you if there are any changes.

In your Bash if you hit an up arrow, then it's going to take you to the previous command.

Now we create 2 new txt files:

```
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        second.txt
```

```
        third.txt
nothing added to commit but untracked files present (use "git add" to track)
```

Untracked = we didn't commit them and if we don't commit them, Git isn't keeping track of them

commit - zatwierdzać

6. Git Workflow

So remember that commiting a file is a 2 step process:
git add . / git add second.txt – add it or get it ready to commit, it's actually called adding it to the staging area. Bucky calls it purgatory, because it's in between your working copy.

3 stages:
working copy (1) > staging area (2) > repository (3)

1) **whenever we create a file it's on our working copy, it's on a local computer**
2) **these are the files that are ready to get commited**
3) **whenever files get commited they go to the repository. When they are there, Git keeps track of those files**

```
Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git add third.txt

Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   third.txt


Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git commit -m "adding 3rd file to the repository"
[master 5d108c0] adding 3rd file to the repository
 1 file changed, 1 insertion(+)
 create mode 100644 third.txt

Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git log
commit 5d108c06184bbd2249ceef6164485baf23d3075d (HEAD -> master)
Author: Miki <mikolaj.myszka@gmail.com>
Date:   Mon Dec 25 21:29:45 2017 +0100

    adding 3rd file to the repository

commit 41e66cbce65c7de571e4e90c5e5840ad5409383b
Author: Miki <mikolaj.myszka@gmail.com>
Date:   Mon Dec 25 21:24:15 2017 +0100

    adding 2nd file to the repository

commit 4fc780b99af5ebff9c7d354d43802d55e0a5917f
Author: Miki <mikolaj.myszka@gmail.com>
Date:   Sat Dec 23 17:59:02 2017 +0100

    This is the first commit
```

7. How to Edit Files

We made edits to first, second and third.txt.

```
Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git status
On branch master
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:   first.txt
        modified:   second.txt
        modified:   third.txt

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

Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git add .

Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   first.txt
        modified:   second.txt
        modified:   third.txt


Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git commit -m "commiting multiple files"
[master b7b943d] commiting multiple files
 3 files changed, 3 insertions(+), 3 deletions(-)
```

8. Viewing the Changes That You Made

1. We change first.txt.

```
Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git status
On branch master
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:   first.txt

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


Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git diff
diff --git a/first.txt b/first.txt
index 6cc2fec..e61be18 100644
--- a/first.txt
+++ b/first.txt
@@ -1 +1 @@
-I am Sam and I love ham. This is the first commit.jk
\ No newline at end of file
+Buck is a cool guy!
\ No newline at end of file
```

git diff – differences between your repository and working copy

Red is in your main repository
Green are the differences or the modifications that you made

2. Now we are going to make modifications to more than one file (looks a little bit different than before).

```
Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git status
On branch master
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:   second.txt
        modified:   third.txt

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


Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git diff
diff --git a/second.txt b/second.txt
index 4074998..c7796c6 100644
--- a/second.txt
+++ b/second.txt
@@ -1 +1 @@
-2nd file.jk
\ No newline at end of file
+2nd file.jk lolilo
\ No newline at end of file
diff --git a/third.txt b/third.txt
index 39134e9..b5c9808 100644
--- a/third.txt
+++ b/third.txt
@@ -1 +1 @@
-3rd file.jk
\ No newline at end of file
+3rd file.jk thied
\ No newline at end of file

Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git add .

Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git commit -m "here are changes to 2 files"
[master cf33c06] here are changes to 2 files
 2 files changed, 2 insertions(+), 2 deletions(-)
```

Sprawdzenie:

```
Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git status
On branch master
nothing to commit, working tree clean

Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git diff
```

9. Comparing the Staging Area with the Repository

Zmieniamy teraz first.txt

```
Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git status
On branch master
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:   first.txt

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

Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git add first.txt

Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git diff
```

diff juz nie pokazuje zmian, bo plik jest już w staging area (zrobilismy add), a nie working copy. How do we compare files in the staging area against the repository?

```
Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git diff --staged
diff --git a/first.txt b/first.txt
index e61be18..5829c17 100644
--- a/first.txt
+++ b/first.txt
@@ -1 +1 @@
-Buck is a cool guy!
\ No newline at end of file
+Buck is a cool guy! Hamblaster.
\ No newline at end of file

Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git commit -m "made some simple changes 9."
[master 9e77e02] made some simple changes 9.
 1 file changed, 1 insertion(+), 1 deletion(-)

Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git status
On branch master
nothing to commit, working tree clean
```

10.

Removes files from git repository and from our computer hdd (deletes working copy).

17. Github

Github is a website where you can pretty much take any project that you're working on and you can publish it for the world to see.
Reason that you would want to do this is because now once you publish it online, anyone can go and they can look at your files, suggest some changes and also what they can do is once they find a project that are like 'ok this looks cool I'm going to work on it', they can just download it all to their local computer, make whatever changes they want to make and then they can submit their changes to this website right here.
So it's pretty much an area or a website where everyone can see your project files, everyone can see the changes that you're making and everyone can collaborate and work on these projects together. Very freaking cool.

So that's what it is, I guess my explanation was a little bit longer than I thought :)
Now anytime you want to use Github whether you're publishing your own projects or just helping out with someone else's, you need a github account.

Making a new public project / repository

'+ ' -> New Repository

'Initialize this repository with a README' leave this option unchecked, because whenever we actually created a repository before, remember that we did '$ git init' and we initialize it right from the command line in Git Bash.We don't want to initialize it this way, because it's already initialized.

'Create Repository'
We now have a public repository.

18. Pushing to a GitHub Repository

A remote is pretty much setting up your connection between your local computer and this public online project, this public server.

```
Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git remote add testpush https://github.com/Mikolaj-Myszka/Test.git
```

Testpush – a nickname for a github project
https... - what url this nickname references

So now we have a connection to our public project

```
Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git remote
testpush
```

Pushing all our files online so everyone can see them.
Whenever you're adding files to your public repository or making any changes, it's called pushing. You're pushing it to the online server. Whenever you're just like browsing through github, you find a cool project and you want to download it or get the files, it's called fetching.

```
Miko³aj@Mikolaj-PC MINGW64 ~/Desktop/Tuna (master)
$ git push -u testpush master
Username for 'https://github.com': Mikolaj-Myszka
Counting objects: 29, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (20/20), done.
Writing objects: 100% (29/29), 14.73 KiB | 1.84 MiB/s, done.
Total 29 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/Mikolaj-Myszka/Test.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'testpush'.
```

The beauty of github is that everybody all around the world can help work on your project.

19.