



**git**



# Introduction to Git

While everyone is settling down, feel free to download Git and create a Github profile!

Git - <https://git-scm.com/>

Github - <https://github.com/>

# Who are we?



Aurora Walker

Lead Product Developer at Pretio, has been using Git for her entire professional career and has **VERY STRONG OPINIONS** about it.



Gyanesh Mishra

Avid Puns and emoji lover! Occasionally also does mobile and backend development at Pretio.

*And thanks to our fabulous volunteer helpers in the crowd!*

# What are we gonna git out of this?

- You'll learn what Git is and how it's different from Github!!!!
- Basic commands/tricks
  - Creating a git repository
  - Making branches
  - Writing **good** commit messages
  - Pushing your code for code review
  - Keeping your branch up-to-date
  - \*How to deal with conflicts (Code conflicts not human)
- Possibly some git puns?

*"It's the git that keeps on giving...."*

# What tools are we using?



Git - The famous version control system

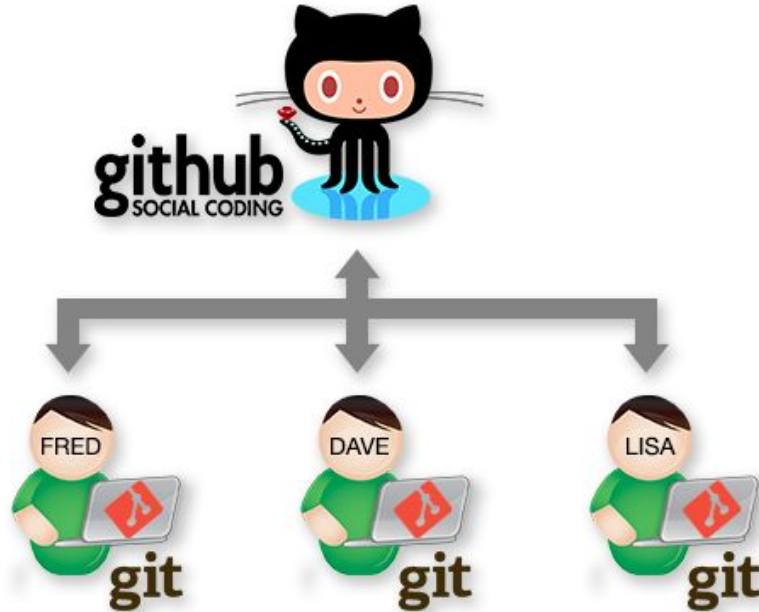
GitHub - To host our remote git repository

Note: There are other tools/services like GitHub (Bitbucket, Gerrit etc). We will be focusing mainly on GitHub because it is popular.

# Is there a difference b/w the two?

Git is the tool, GitHub is the service!










GitHub lets you host multiple git repositories for team collaboration



# What is git? Why should i care?



Basically it keeps track of file changes in your project so you don't end up doing this:

Name	
	120525_document_updated.txt
	120604_document.txt
	120605_document_amended.txt
	120605_document_John.txt
	120605_document_latest.txt
	120605_document_latestcopy.txt
	120605_document.txt
	1200602_document.txt
	document_meeting.txt

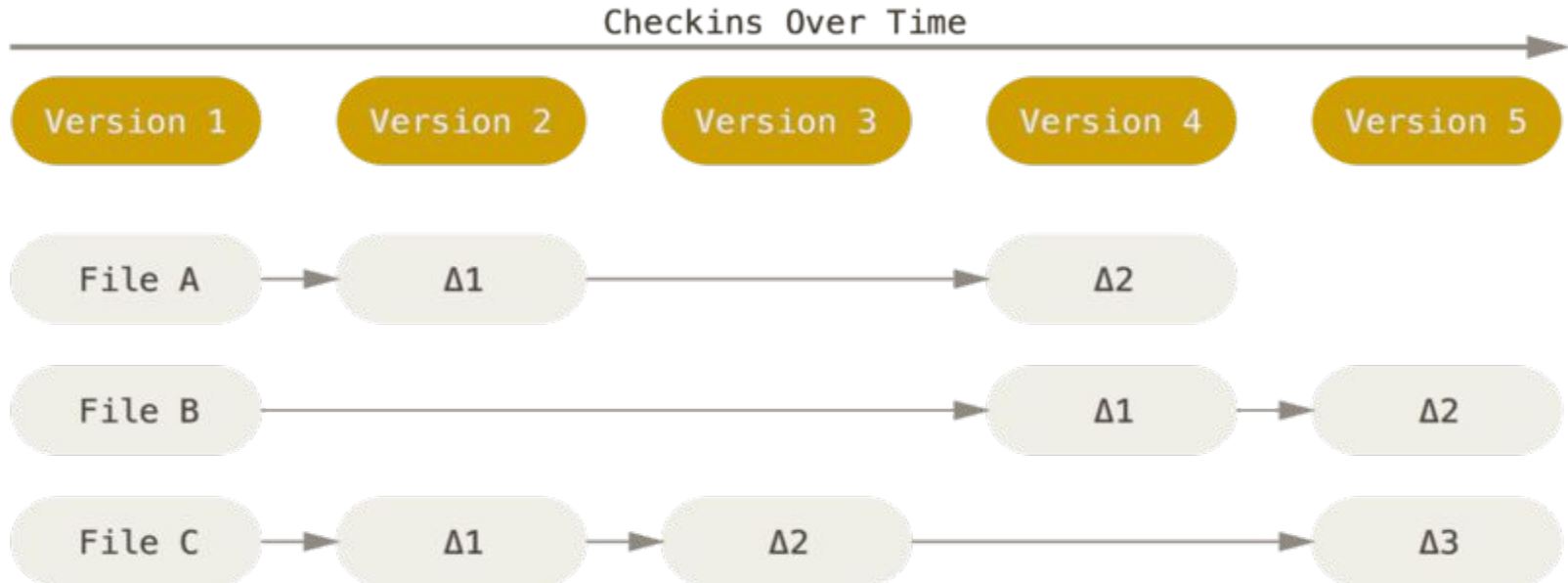


# How is it different from others?



There are multiple systems out there that achieve the same goal like CVS, Subversion, Perforce, Bazaar etc.

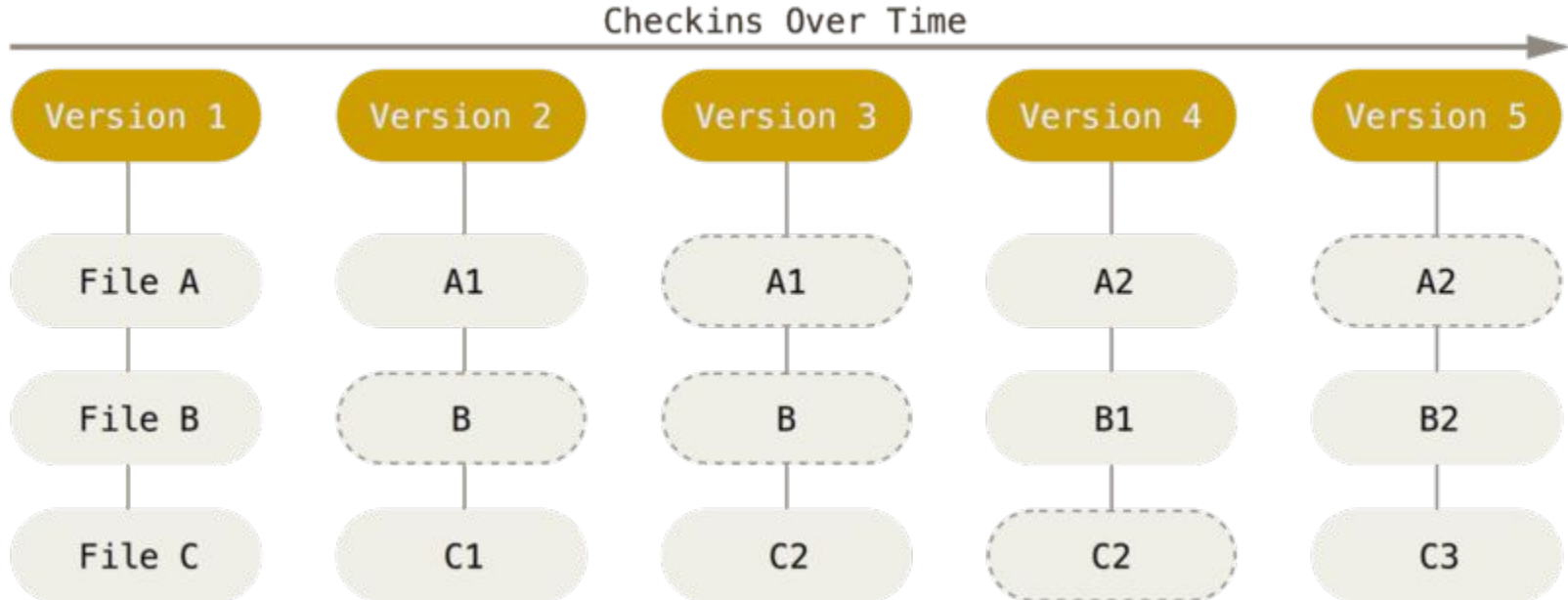
These systems track individual files:



# How is it different from others?



Whereas git tracks your whole project and keeps a snapshot of the whole project state at the time of commit.





# Let's create our Repo!



Log into github.com and click on start a project!

## Learn Git and GitHub without any code!

Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.

[Read the guide](#)

[Start a project](#)

# Let's create our Repo!



- Enter the repository name and description
- Make it a **Public** repository
- Check the box which says **“Initialize this repository with a README”**
- Click on Create repository!

## Create a new repository

A repository contains all the files for your project, including the revision history.

Owner



Repository name

startupslam2016



Great repository names are short and memorable. Need inspiration? How about **jubilant-potato**.

Description (optional)



**Public**

Anyone can see this repository. You choose who can commit.



**Private**

You choose who can see and commit to this repository.

☒ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None**

Add a license: **None**



Create repository

# Let's create our Repo!

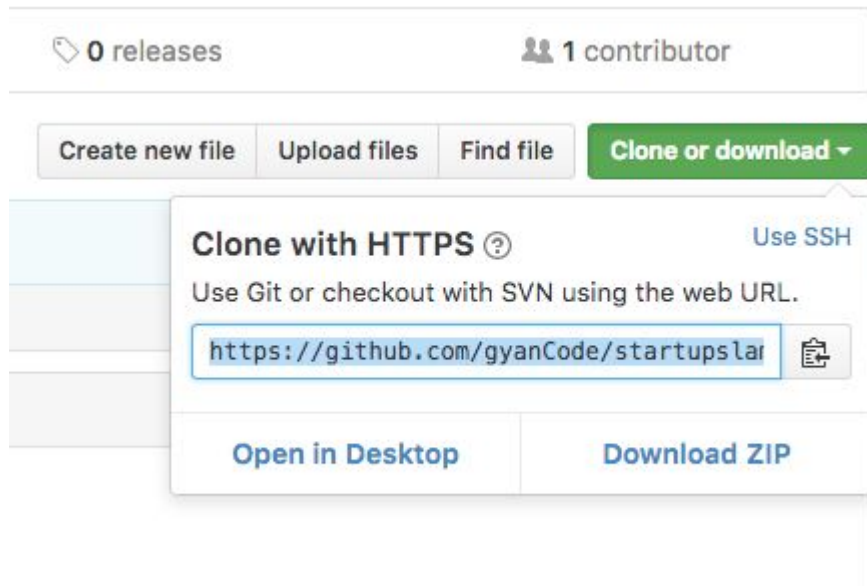


- Copy the Web URL under the “Clone or download” button
- Open a terminal window and navigate to your desired folder
- Type the following command to clone the repo:  

```
> git clone <URL>
```
- List the repo name  

```
> ls
```
- Navigate to the repo  

```
> cd <Repo name>
```



# Let's Git Started!!!!



Whenever in doubt!

```
> git status
```

Git status tells you the 'status' of your repository. It will let you know if you have files that have been changed, if you are in sync with the master branch on the server, and much more!

Use it often! Just know that it doesn't work in slack.



thanks for confirming @cari !!



**aurora** 11:40 AM

git status



**aurora** 2:04 PM

git status



# Let's Git Started!!!!



We will be going over different commands that will help you learn the usage of git.  
We have a zip file with example content for us to edit and work with.

## **First step:**

Download the zip at the link below:

URL - <https://goo.gl/7l8Sdg>

## **Second step:**

Extract the zip and copy the contents to your local repo folder

# CAUTION



Before we start making any changes! Remember one golden rule:

**\* NEVER COMMIT TO MASTER BRANCH DIRECTLY! \***

When you work on a team, it is best practice to make 'Pull Requests' and give everyone a chance to review and collaborate on the code *before* it is added to the Master branch.

So let's make a branch for our changes and then start making changes.!!!

```
> git checkout -b <branchName> --track origin/master
```

# What did I just do?



`git : use git`

`checkout : checkout (Work on) a branch`

`-b : create the branch`

`<branchName> : give the new branch the following name`

`--track origin/master : follow the master branch on the server for changes`

`* Origin refers to anything that's on the server`

# Let's make some changes!!

Open “file1.txt” and add a few lines to it. And let's see what is the status of our repo.

```
> git status
```

Now your changes are in the “staged” version, but not ready to commit.

First, we need to tell git what changes we want in our commit by using

```
> git add .
```

\* You can also use `git add <filename1> <filename2>` and so on if you have multiple files and would like to cherry-pick them.

```
> git status
```

You can see that your changes are now ready to be committed, which we do by:

```
> git commit -m "<Descriptive commit message>"
```



# What did I just do?



`git : use git`

`commit : commit the added changes`

`-m "<Descriptive commit message>" : with the following "message"`

\* The `-m` flag is optional. I recommend using it if you don't like dealing with `vi` and like saving time.

# Your commit message



A good commit message is a short description of what changes happened in this commit. A person should be able to read a commit message and have an idea of what happened without having to read the code.

```
git commit -m "Adds the password reset feature"  
git commit -m "Adds tests for account creation"
```



A bad commit message is too generic, and doesn't really tell you what happened in the commit.

```
git commit -m "bugfix"  
git commit -m "added tests"
```



# Wait! It says i'm ahead?



Let's see what the status of our repo is now.

```
> git status
```

```
python ... pretio@worker:~_repo bash
(pretio)[aurora@Auroras-MacBook-Pro pretio (PRET_1718)]$ git status
On branch PRET_1718
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working directory clean
(pretio)[aurora@Auroras-MacBook-Pro pretio (PRET_1718)]$
```

Your local branch now has one extra commit than the server doesn't have yet.

*Right now, your changes are trapped on your computer!*

We will want to get them up to the server for everyone to see.

# Time to Sync up!



Now that we're happy with our local changes. It's time to push them to the central server (on GitHub) to get code review and feedback.

To push your branch:

```
> git push origin <branchName>:<branchName>
```

# What did I just do?



`git : use git`

`push origin : push the local changes to the remote repository`


`<branchName> (local) : <branchName> (server) : name of the branch`

\* the branch will be created if doesn't exist and it will give you a warning if it does

Now we'll use the GitHub web interface to open up a pull request, get code review feedback from our teammates and merge our code so everyone can have updated code in their local repository.

Since we just pushed our code, when you go to the repository on GitHub, you'll see the following:

Your recently pushed branches:


 test (less than a minute ago)


 Compare & pull request


Now there is an copy of the branch from your local computer on the server!

## Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

 base: master ... compare: test ✓ Able to merge. These branches can be automatically merged.



Updated description 


Write

Preview


AA ▾ B i “ <> 🔗 ☰ ☷ ☹ ↶ @ ★

- More description
- Possibly why it was needed and what it does?
- if I added/removed something else etc.
- Link to a ticket if exists etc.


Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

 Styling with Markdown is supported


Create pull request

Labels 

None yet

Milestone 

No milestone

Assignees 

No one—assign yourself

Once you open a Pull request, members in your organization who has access to the repo can go and add their reviews/comments with in regard to your code.

Code review cycle:

Make changes -> Add -> Commit -> Push.

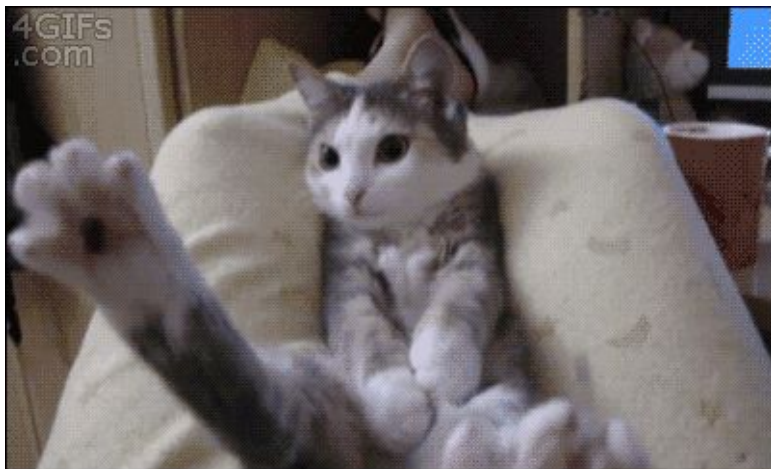
Repeat till team-mates happy.

Once that happens you just merge your code by clicking on the button below:





# Tiny break!



For cat lovers!!



For dog lovers!!

# Time to Sync up



Alright! Your shiny new feature is now merged on the remote server.

Let's see the status of the local repo (*think about what branch you are on!*):

```
> git status
```

List all the branches you have on your local repo (the -vv means 'verbose!):

```
> git branch -vv
```

Let's switch to master by doing :

```
> git checkout master
```

Aaaaaandddd..... The status

```
> git status
```

# What did I just do?



`git` : use git

`checkout` : work on

`master` : master branch

\*master branch on your local is your local copy of the origin/master branch, which is on the GitHub server! Remember: 'origin' means on the server.

Hot Tip!



To switch to any existing branch on your local computer at any time, use the command: `git checkout <BranchName>`

# Time to Sync up



```
On branch master
Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)

nothing to commit, working directory clean
```

The master branch on local is out of sync, to get it up to date with origin, enter the following command:

```
> git pull --rebase
```

```
First, rewinding head to replay your work on top of it...
Fast-forwarded master to 957e144f93fbb4c8a62e15ec5effe0fc83f16399.
(pretio)[aurora@Auroras-MacBook-Pro pretio (master)]$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean
(pretio)[aurora@Auroras-MacBook-Pro pretio (master)]$
```

# What did I just do?



`git` : use git

`pull` : grab the changes from the remote repository (origin) and update my local one (secretly a combination of two other git commands `fetch` and `merge`)

`--rebase` : Replay all commits and put my changes on top of it

The controversial '`--rebase`' flag:

- Keeps the commit history clean by rewriting it
- One of those things Aurora has strong opinions on

# Why did I just do that?



```
> git pull --rebase
```

Is a great command to run periodically while you are working on a branch. If you spend a couple of days working on a feature (or if your teammates commit a lot of code!) this makes sure that you are always pulling their changes down into what you are working on. This command gets you any new code that has been added to the master branch since you started working.

Stay up to date! Pull and Rebase today!

*“Git pull a day keeps the conflicts away”*

# Conflicts!!!!



You and your buddy edited the same file!! Git gets confused while rebasing and needs a human to sort it out!




# Resolving conflicts!



Let's open the file that is conflicted:

```
> vi conflictedFile.txt
```

Edit the file as needed! Delete and move things around, using your human brain  to decide how the file should look.

When you are done, add the changes, so git knows you have dealt with the conflicts:

```
> git add conflictedFile.txt
```

Tell git everything is fine and continue with the rebase:

```
> git rebase --continue
```



# Review All the Steps



Repeat!

- Make a new branch
- Write code/content/whatever into some kind of work unit on that branch
- Commit your changes
- Push your branch to the server
- Make a Pull Request on the GitHub server
- Get feedback from teammates
- Merge your branch to Master when it gets approved 🙌
- Pull down you changes
- Start again!

# Bonus Commands



Delete a local branch that has been merged to master and you don't need anymore.

```
> git branch -d <BranchName>
```

See the history of commits on the branch you are currently on (hit `q` to exit when done!):

```
> git log  
> git reflog
```

If you accidentally added a file that you *do not* want in your commit, you can remove it with:

```
> git rm <filename> --no-cache
```

# Bonus Commands: STASH



If you have changes you want to 'hide' instead of commit, use the command:

```
> git stash
```

To see all the things you have stashed away use:

```
> git stash list
```

To take the top item you have stashed, and bring it back:

```
> git stash pop
```

Delete the top item you have stashed:

```
> git stash drop
```

For more stash commands see:

```
> git stash help
```

# References



[https://backlogtool.com/git-guide/en/intro/intro1\\_1.html](https://backlogtool.com/git-guide/en/intro/intro1_1.html)

<https://git-scm.com/book/en/v2/Getting-Started-Git-Basics>

<http://emojipedia.org/>

<http://ohshitgit.com/>

<https://github.com/EugeneKay/git-jokes/blob/lulz/Jokes.txt>