# Getting to know Git

**Powered by: IEEE CS RVCE Chapter and IEEE CS Bangalore Chapter**


## ANAND JAGADEESH

**Software Engineer 1, PowerMax Replication And Cloud, Dell EMC (Dell Technologies)**

**Secretary, IEEE Computer Society Bangalore Chapter**


IEEE COMPUTER SOCIETY Bangalore Chapter

IEEE Advancing Technology for Humanity

# About me

- Anand Jagadeesh / Anand J. / AJ
- Secretary, IEEE Computer Society Bangalore Chapter and IEEE CS Members for over 7 years
- Software Engineer 1 in PowerMax Remote Replication team of PowerMax Replication and Cloud, Dell EMC (Dell Technologies)
- A Git user – Wouldn't call myself an expert – Just a learner
- FOSS Enthusiast

# In this workshop, you will learn,

- what Git is and how to install and configure it
- the basic commands in Git – init, add, commit, push
- remote, clone, pull, fetch, merge, reset, revert, stash, clean, rm  commands in Git
- branching in git and merging branches
- hosting a personal website or project website using GitHub pages
- GitKraken, GitHub Desktop and Git GUI tool

# Please Note!

# Things to remember and conventions used in this slide deck – Please read this!

- For commands:
  - Anything in square brackets ( [ ] ) are optional parameters or options to the command. If you do not specify them, the default parameters will be used. Do not put the square brackets ( [ ] ) in actual commands.
  - Anything in angle brackets ( < > ) are variables. For example <name> means you need to specify your name and <email> means specifying email. Do not put the angle brackets ( < > ) in actual commands.
  - Anywhere in the command where an ellipsis ( … ) is given it meand more such parameters or options to be specified. Do not put the ellipsis ( … ) in actual commands.
  - Anywhere there is a commit number followed by 2 dots ( .. ) then another commit number it is required. You can see these when you see diff between two commits, etc.
  - The vertical slash ( | ) in command options or parameters means “OR”. For example origin|<remoterepopointer> means you need to specify either “origin” or “<remoterepopointer>”

# Things to check after you finish the main topics!
# Don’t Forget!

- Appendix A: Why Anand Jagadeesh suggest SSH over HTTPS as the protocol for PUSH and FETCH?
- Appendix B: How to setup SSH? The step-by-step guide
- Appendix C: Anand’s references for this workshop

# The Git Story

# The Git Story

- How many documents, with the same name do you have?

  name_name name_new name_name_1 name_new_newer

- VERSION CONTROL SYSTEMS or VCS

- Types:
  Local Only (FOS or Propr.): RCS – 1982, PVCS – 1985
  Client-Server (FOS or Propr.): CVS – 1986 and 1990(in C), SCM – 1970s
  Distributed (FOS or Propr.): Git – 2005, Plastic SCM – 2006 (Source: WikiPedia)

# The Git Story

- Created by the man who created “Linux”
- The name?
  - *unpleasant person*
  - "the stupid content tracker“
  - Pronounceable – not a UNIX command
- 2005/07/11 – 0.99
- 2020/07/27 – 2.28.0
- Design is inspired by BitKeeper and Monotone
- Distributed
- HTTP/FTP/SSH

# And the Git is

- Small and Fast - Almost all operations are performed locally
- Built to work on the Linux kernel – Can handle large code
- Its written in C – Reduce overheads during runtime
- Distributed – Multiple Backups Available Always
- Free and Open Source

# And the Git can

- Support Multiple workflows:
  - Subversion-Style Workflow – Shared Repository – Centralized
  - Integration Manager Workflow – Single person committing to “Blessed Repository”
  - Dictator and Lieutenants Workflow - Single person committing to “Blessed Repository” but multiple Lieutenants are in charge of subsystems
- Assure the right data is stored and available
- Has an intermediate storage area before permanently keeping the changes
- Can do Branching and Merging

# Let's Git It Started

# Get Git

- GitHub for Windows | git-scm.com/downloads
- GitHub for Mac | `git --version`
- Git distributions for Linux and POSIX systems are available on the official Git SCM web site
- Linux Users can run `[sudo] apt get install git-all`
- I would suggest using SSH keys but let's skip for now

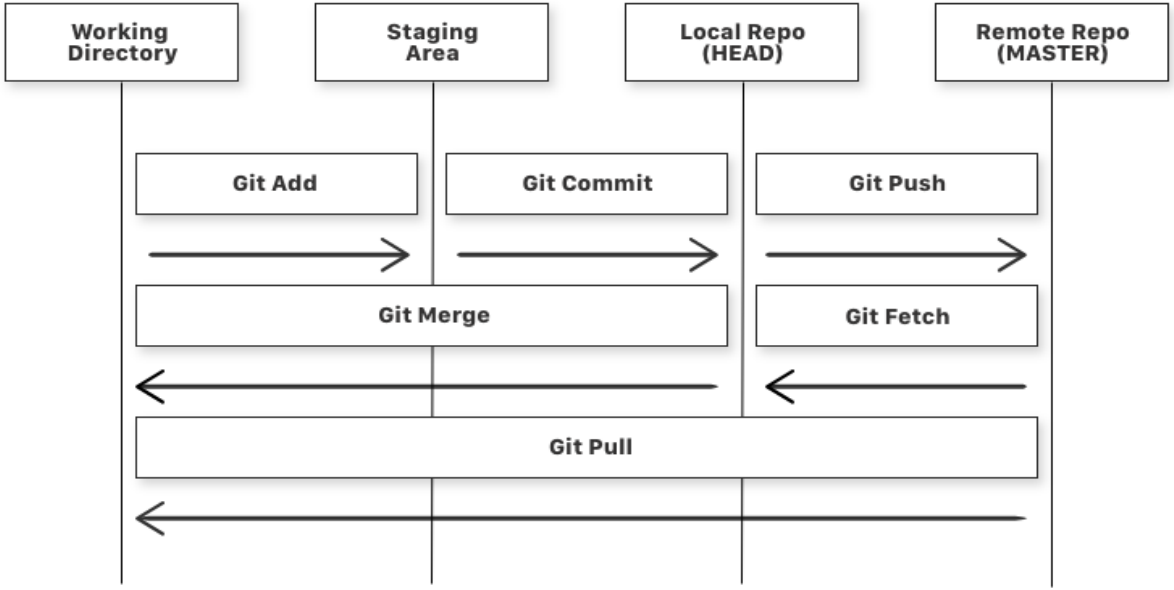
# Config Git

- Telling git who you are, is important.
- Method 1: You can edit the git config file – Skipping as it's not easy for beginners
- Run the following:
  - `git config --global user.name "<firstname><space><lastname>"`
  - `git config --global user.email "<email>"`
  - Optional: `git config --global color.ui auto` | Enables helpful colorization of command line output
- `git config --global –list` | see what you just set
- https://git-scm.com/book/en/v2/Customizing-Git-Git-Configuration

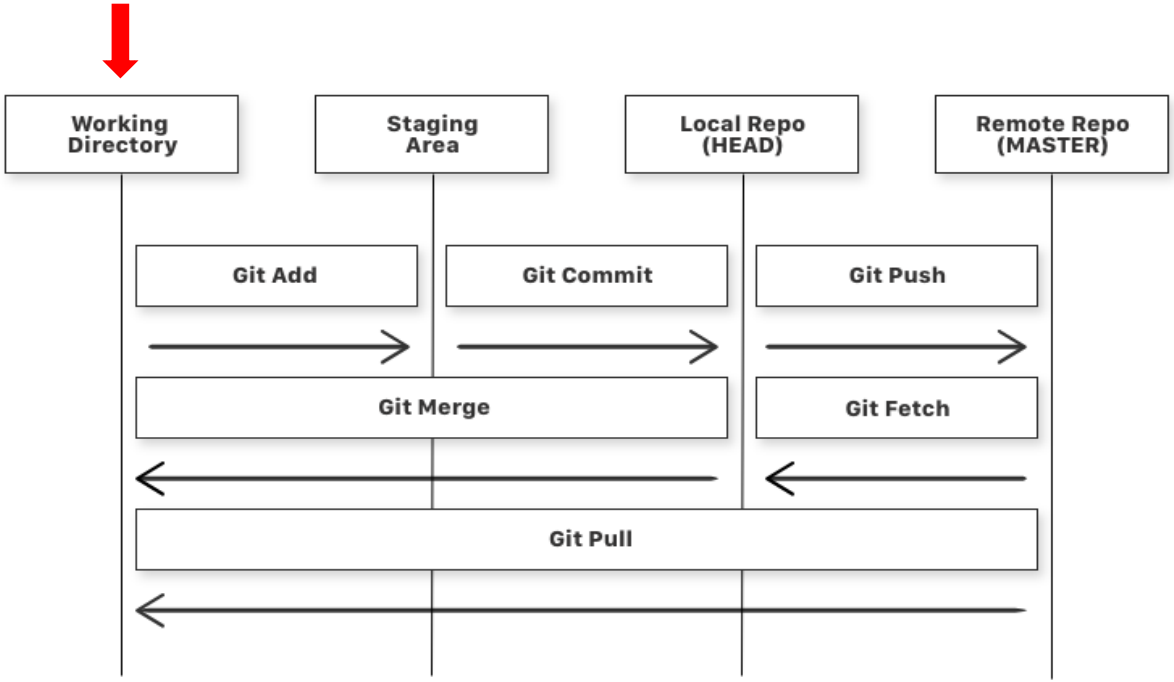
# The First Steps

- Create a directory/folder
- Initializing a repo, run: `git init`
- Creates the underlying metadata required
- Then, add file(s) to the directory/folder
- Now files are in the working directory
- To see the status, run: `git status`

# The First Steps



# The First Steps



# Stage It

Staging is telling git that you would like these changes to be permanent. This is an intermediate stage, changes will not be permanent after this.

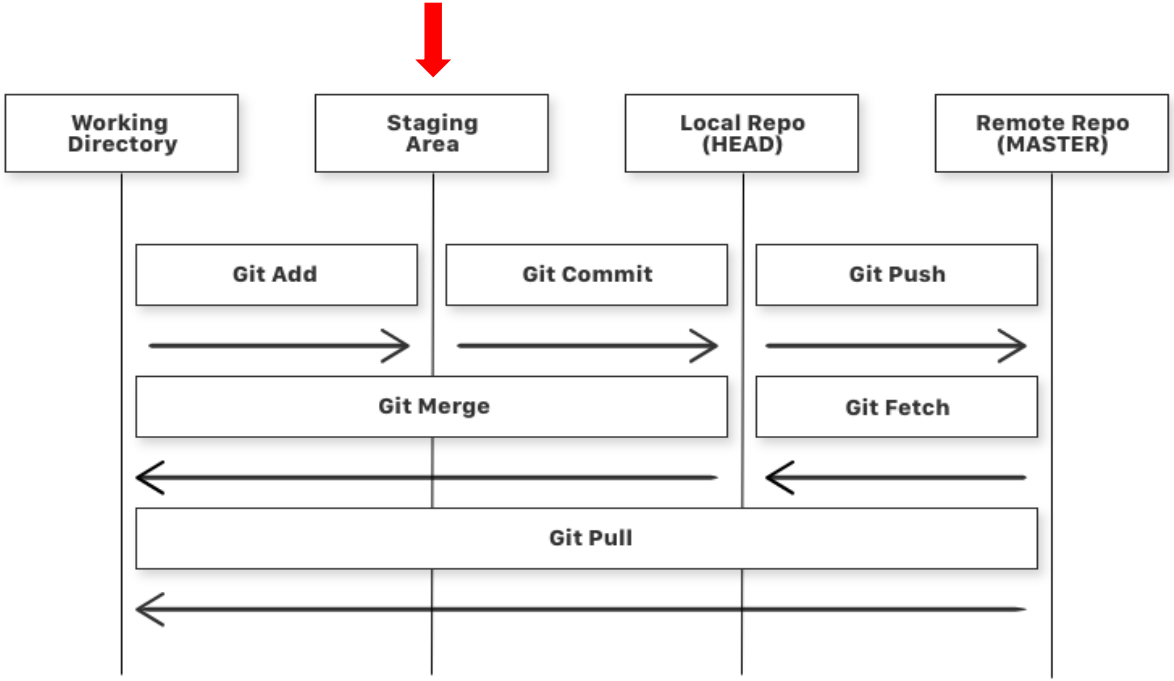
In terminal, enter

- `git add <filename>` ----> Stages the filename

Or

- `git add .` ----> Stages all changed files in the WD

# Stage It



# Commit It

- Now files are staged
- We just made it ready to be made permanent – like expression of interest
- To make it permanent, we need to commit the changes

```
git commit [-a] -m “my message”
```

- Want to fix typos or piggy-back changes on last commit?

```
git commit --amend
```

- If you are using `–amend` you can also give `git commit [--no-edit]` -------> `--no-edit` will not open up editor for commit message

- Now **HEAD** (a pointer used by git to point to last commit on the branch) is the new commit ----- First commit is the root-commit ----- **Git uses a tree structure for your commits**

# To See All Commits or Contents of a Commit

- To see all commits, run

```
git log
```

- It lists all commits. You have to type ‘q’ to exit
- You can filter using --author=<name> to filter on an author
- Other filters also available

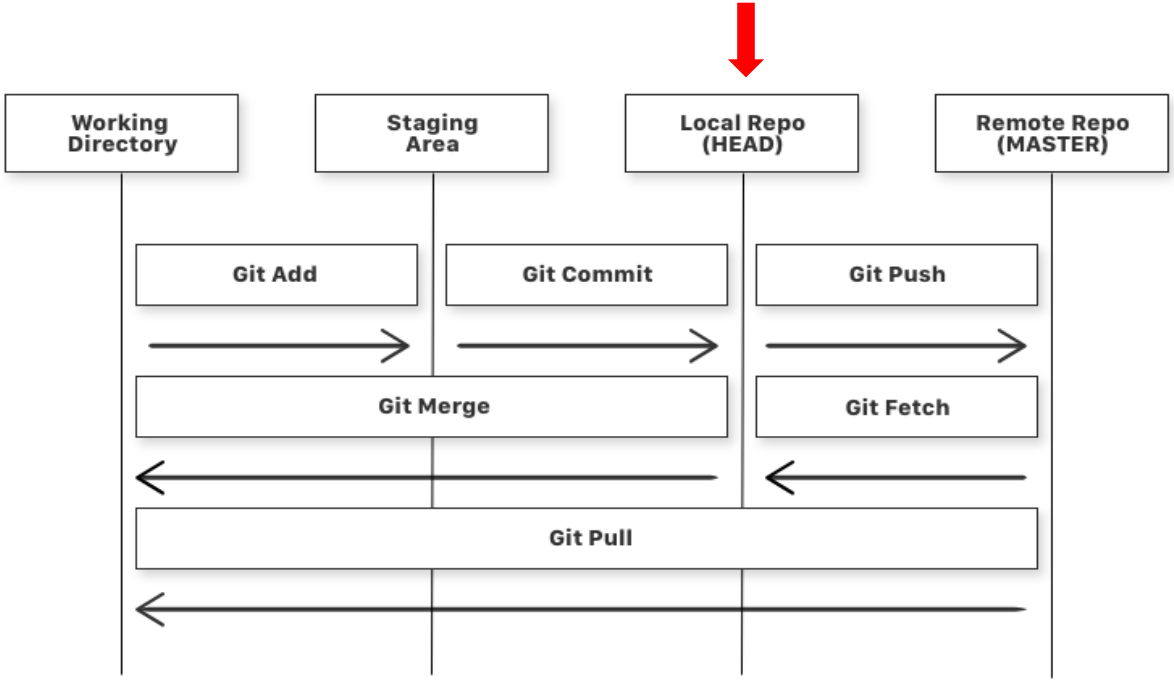
- To see contents of a commit. Find the commit from git log and run

```
git show <commit#>|<first_7_chars_of_commit#>
```

- To see contents of HEAD, run `git show HEAD`
- To see contents of last to last commit, that is, HEAD’s parent, run `git show HEAD~`

To see file names that were changed in a commit only, use option `--name-only`

# Commit It



# Where’s my Remote?

# Importance of Remote Repository

- Now everything is in the local space – Just that changes are tracked

- **What if we loose the computer?**

Or

- **Accidentally delete the folder?**

- Boom! You will be an inclined plane helically wound around an axis! (For those who didn't understand this phrase, I'm sorry)

- So, the solution is a "**Remote Repository**" Tadaaa!!!!

# Remote Repository

- Could be on-premise or enterprise-hosted/leased or hosted elsewhere or cloud, mostly privately hosted
- Or can use the services or products like GitHub, GitLab, BitBucket, etc.
- Let's try GitHub in these examples

# Creating an Account on GitHub

- Create an Account on GitHub at https://github.com/join

- Once created, just create a new repository but, do not initialize it with a readme or anything.

- Once repository is created, copy its URL – I suggest using SSH (see appendices for details on why)

- That's your remote repository – Now let's connect our local to remote

# Let's Connect Repos and Push!

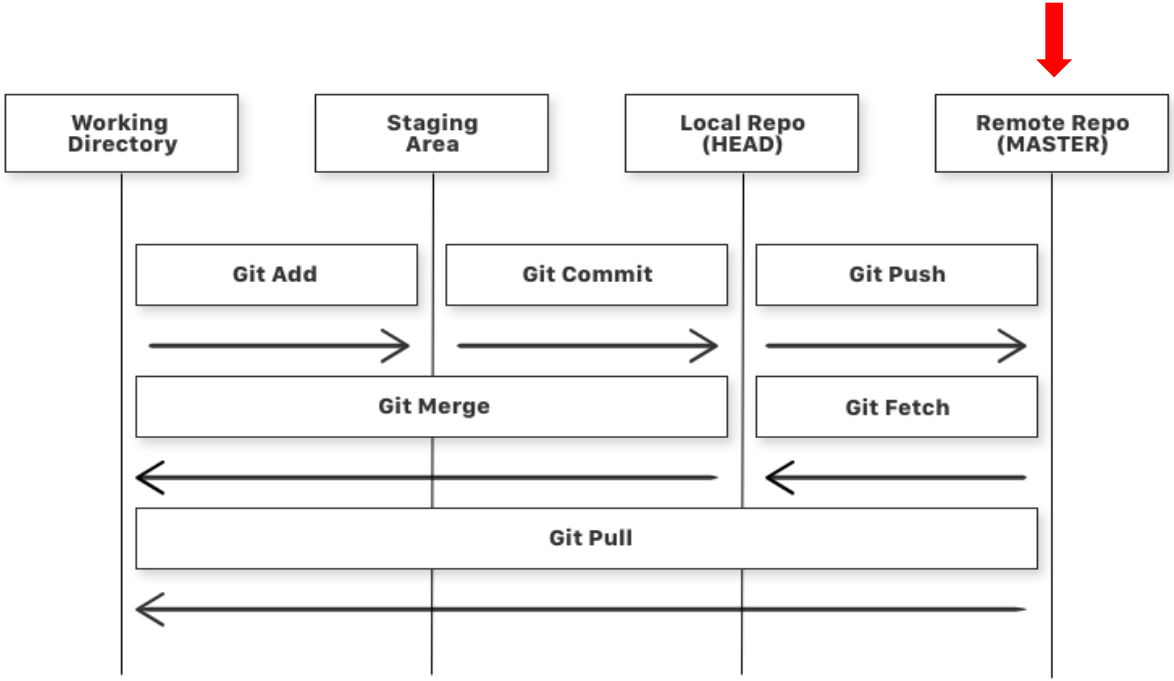
- Back to the terminal!

```
git remote add origin <the_copied_url>
```

```
If SSH, generate SSH key and add it to GitHub(see appendices)
```

- Verify using: `git remote -v`
- Push to remote: `git push [origin] [<branch>]`
- KaBoOm! Your Changes are in Remote now!

# Push It

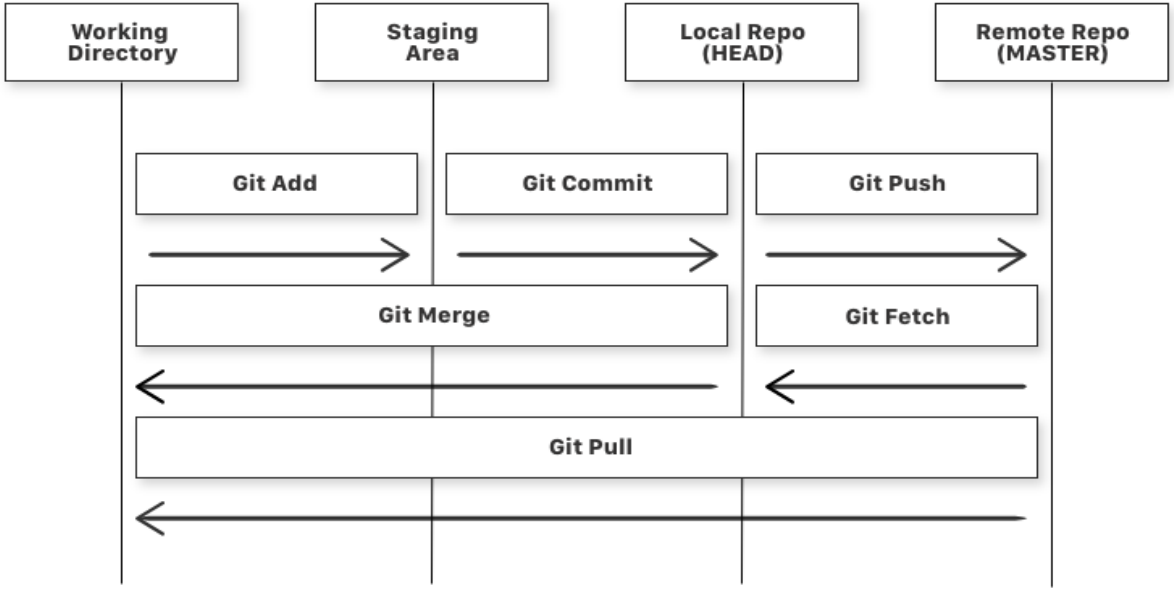


# Some Things to Remember

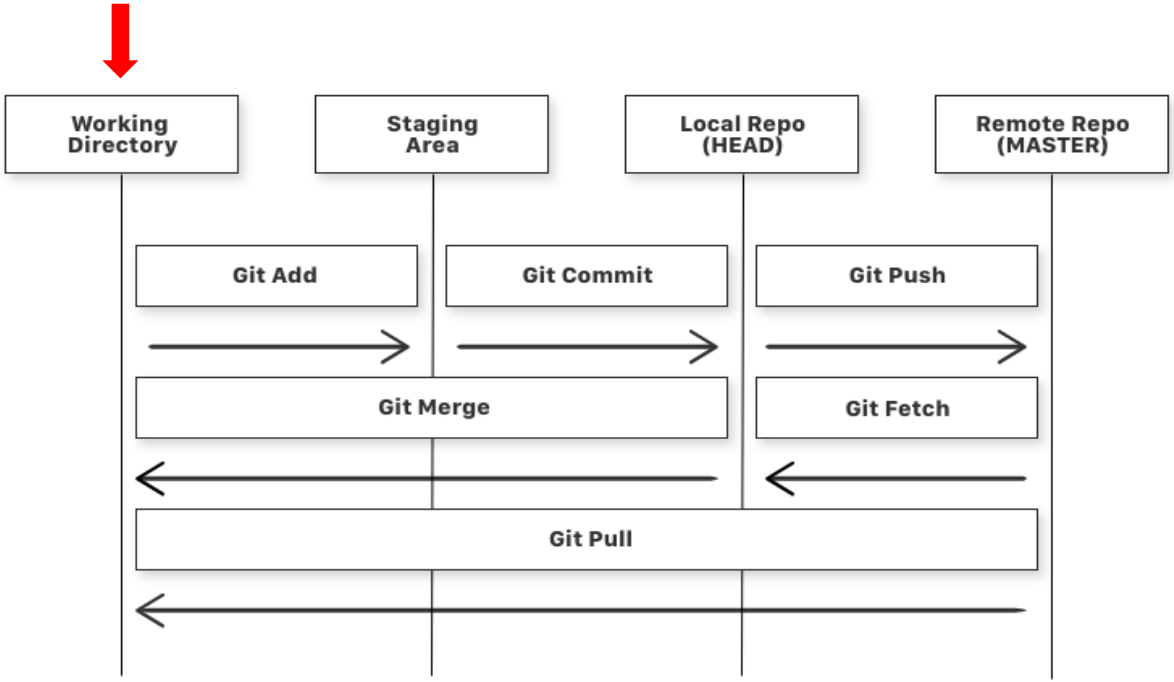
- **HEAD**
  - Pointer to latest commit
  - On local repo – Points to latest commit
  - On Remote Repo - Points to latest commit
- **origin**
  - Pointer to remote repository – Default – You can use other names
- **git show HEAD**
  - **Shows last commit contents**

# A Quick Revision

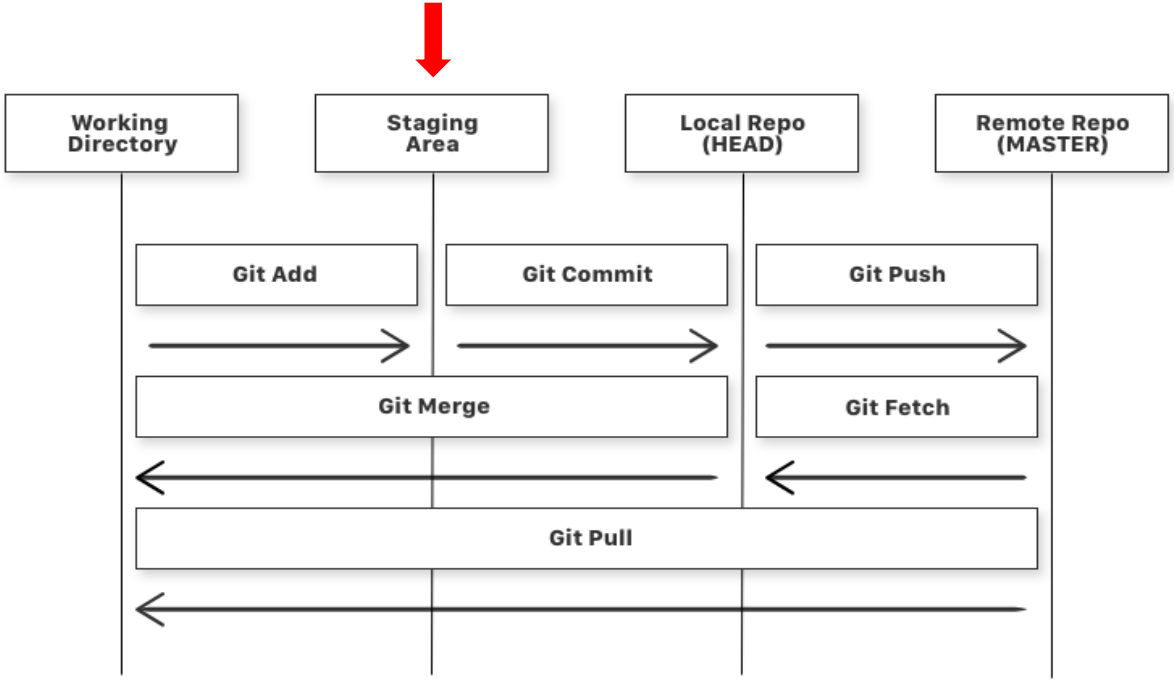
# The First Steps



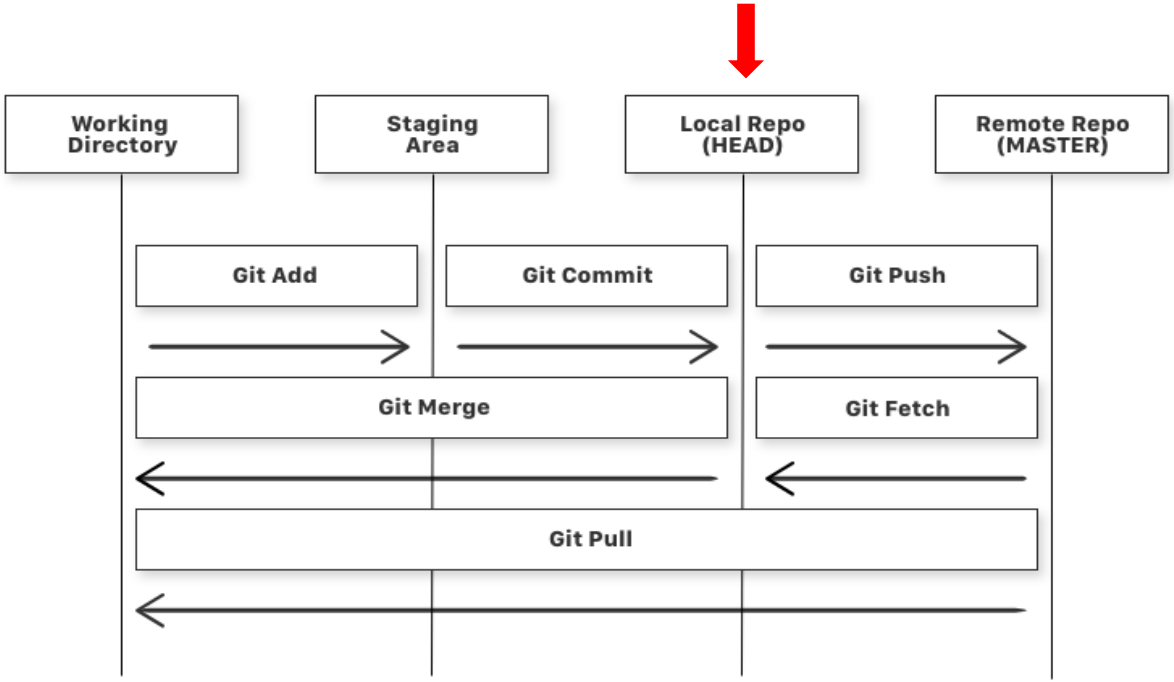
# The First Steps



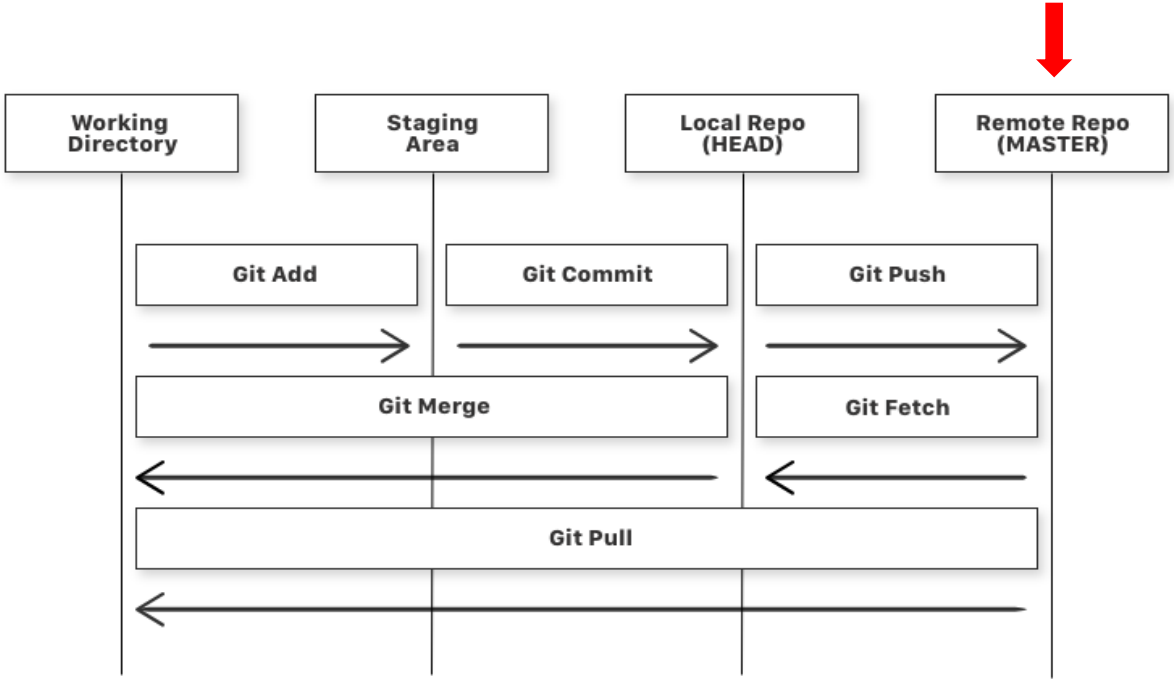
# Stage It



# Commit It



# Push It



# Clone, Pull, Repeat!

# Cloning a Repo

- Back to the terminal!
- It's bringing a repository from the Remote to Local to work on
- Create a repository on GitHub, initialize with a README and copy the link to repo (I prefer SSH)
- `git clone <link_you_copied> [<folder_you_created>]`
- If folder is not specified, it will clone to a folder/directory with **same name as the repo**, in your current folder/directory

# Pull from Remote

- Consider the remote have changes pushed by someone else. You need to get them! Or you edited files on GitHub and you need to get them to local. You need to do a **PULL** (Its Fetch(bringing the changes) plus Merge(applying the changes))

- `git pull [origin|<remoterepopointer>] [<branchname>]`

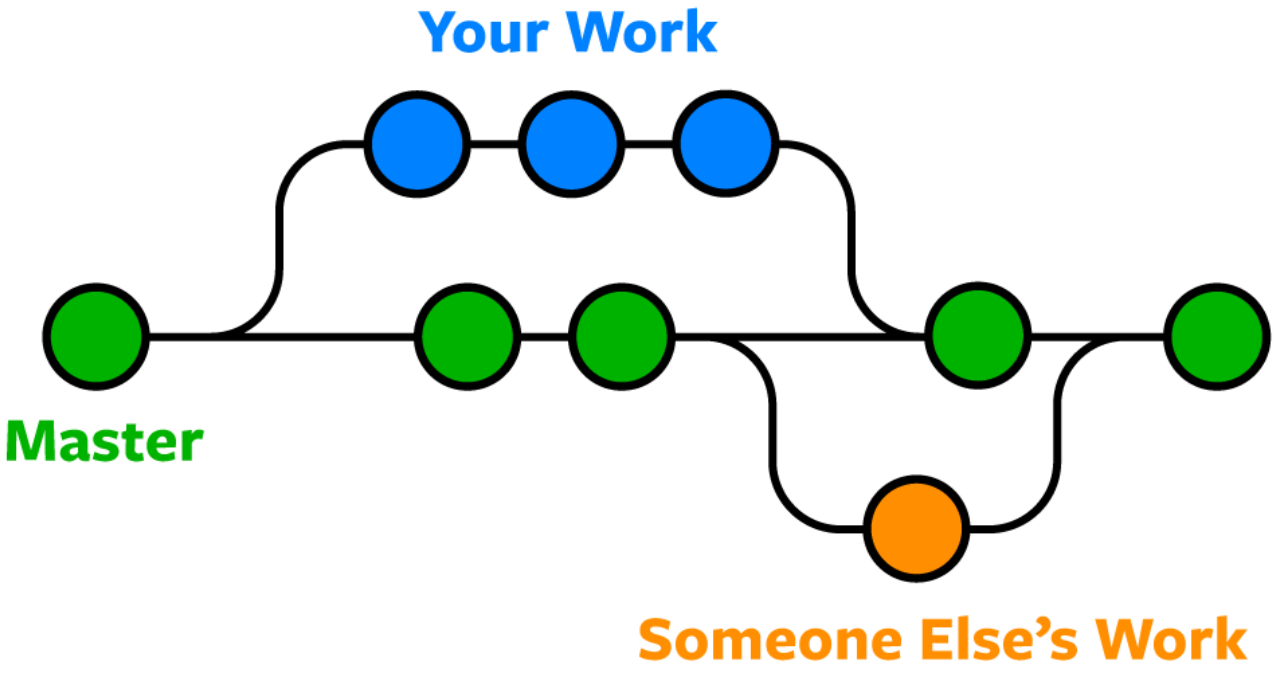
- If you do not specify origin or <branchname>, it takes the defaults. The default branch name is current branch you are on. See the topic Branching in Git in future slides.

- You need to set the remote to do this. See section in previous slides on remote repository

- Conflicts during a pull needs to be resolved just like how we resolve conflicts during a merge. See future slides on Merging

# Branching out!

# Branching in Git



# Branching in Git

- Back to the terminal!
- `git checkout -b <branch_name> ----> Creates a branch`
- `git branch ----> See all branches`
- `git checkout <branch_name> ----> Switch to branch`
- `git branch -D <branch_name> ----> Delete a branch`
- `git push origin --delete <branch_name> ----> Delete a branch in the remote repo`

# Merge It!

# Merging Branches

- Back to the terminal!
- `git merge <branchname> ----> merges the contents of branch branchname to the branch currently in`
- `git merge --abort  ----> Aborts the current merge`
- If the merge has no conflicts, it does auto-merge but if there are merge conflicts, meaning, changes to same line coming from different branches, it wants us to resolve the conflicts
- See next slide for steps

# Merging Branches

- You need to do conflicts resolution for each file
- Run git status. The conflicted files will be listed as:

both modified:  <filename>

- Open each of those files

- Find the conflict markers that looks like:

<<<<<<< HEAD

<some changes>

=======

<other changes>

>>>>>>> <commit#>

# Merging Branches

- Whatever coming between the <<<<<<< HEAD and ======= is the changes that's there in our local branch, the one we are merging into
- Whatever coming between the ======= and >>>>>>> <commit#> is the changes coming from the branch we are merging from
- You need to manually select the lines and changes you want to keep and then remove the <<<<<<< HEAD , ======= and >>>>>>> <commit#> markers and save the file
- Once that's done, do a `git add .` or `git add <filename>` for each of the files and just give `git commit [--no-edit]`
- `--no-edit` will not open up editor for commit message

# Its #PR Time!

# Pull Requests on GitHub

- You can raise a Pull Request on a main branch from any other branch
- Demo Time!
- Why Pull Requests?

# Stash - Unstash

# Stash and Unstash!

- Used to temporarily store changes when we need to switch contexts without loosing data!
- Very handy when working with multiple branches and multiple projects!
- Can have multiple stashes
- Each are numbered – Can be referred with it

# Stash and Unstash!

- `git stash ----->` Stashes the current changes in the WD
- `git stash apply [<stash>] ----->` Applied the top stash or specified stash on WD
- `git stash drop [<stash>] ----->` Remove the top stash or specified stash on WD
- `git stash list ----->` Lists all the stashes
- `git stash show [-p] ----->` Shows the changes in the stash
- `git stash branch <branchname> ----->` created a branch from the changes that were going to be stashed

# Fix Mistakes

# Oops I put mistakes in The Last Commit!

- Mistakes can always be corrected
- Method one: Resetting a branch to a commit/HEAD:
- `git reset [<mode>] [<commit#>|HEAD]`
  - `--hard --------> Staging index and WD reset ----> Beware! Changes will be gone! And irreversible!`
  - `--mixed --------> Changes from staging index moved to WD – Default option`
  - `--soft --------> Staging index and WD not changed`

# Oops I put mistakes in The Last Commit!

- Mistakes can always be corrected

- Method two: Reverting a commit:
- **Creates a neutralizing commit for the commit we are reverting**

- `git revert [-e|--edit|--no-edit|-n|--no-commit] <commit#>`
  - `-e or –edit will open up editor to give commit message`
  - `--no-edit will not open the editor for commit message – default message is used`
  - `--no-commit will not create a commit but neutralizes the changes and the new changes are in your WD you need to stage and commit`

# Quick Clean-Ups Before Pushing

# Quick Clean-ups

- `git clean [-n] [-f|--force]`
  - To cleanup the WD of unused and untracked files
  - -n will do a dry-run – meaning it won't cleanup but does an analysis of what all will be cleaned up and tell you the result!
  - -f or --force will do a force cleanup

- `git rm <path_to_file>… [-f|--force][-n|--dry-run][-r][--cached][--quiet]`
  - deletes a file or files – can be fixed with reset command
  - updates the staging index and the WD – Then you can commit it
  - -f or --force force removes the file(s)
  - -n or --dry-run will do a dry-run – meaning it won't remove but does an analysis of what all will be removed and tell you
  - -r does recursive on folders
  - --cached removes cached/staged changes
  - --quiet will not print a line of output for each deleted file

# Hosting Web Pages on GitHub Using GitHub Pages

# Let's see all this in action – How about a Personal Website using GitHub Pages?

- Step 1: Create a repo on GitHub with name
  <username>.github.io
- Step 2: Add the required HTML files and supporting files to the repo by either uploading on the GitHub web interface or clone the repo, add the files and push them
- Step 3: Go to Settings tab on the repo and find the section "GitHub Pages"
- Step 4: You can see that already the site is available at https://<username>.github.io
- You can change the branch to generate the page from or put a custom domain there to point to this site

# Let's see all this in action – Serving webpages from other repos using GitHub Pages as subpage

- Step 1: Create a repo on GitHub
- Step 2: Add the required HTML files and supporting files to the repo by either uploading on the GitHub web interface or clone the repo, add the files and push them
- Step 3: Go to Settings tab on the repo and find the section "GitHub Pages"
- Step 4: Select the branch to generate the page from or put a custom domain there to point to this site and hit save
- You can see that the link to that webpage is: https://<username>.github.io/<reponame>

# Git Operations using GUI

# INTRODUCING MYSELF AGAIN

**Anand Jagadeesh**
@_anandjagadeesh
#gitkrakenambassador

*I am a GitKraken Ambassador,* ***not a paid employee of GitKraken by Axosoft.***

# GUI Tools!

- Suggest -----> GitKraken: https://sl.anandj.xyz/GK
  - Has a cool UI
  - Free Pro account with GitHub Student Packs
  - Also have tools: Glo Boards and Timelines – Helps your projects
- GitHub Desktop: https://desktop.github.com/
- Git GUI: Right Click Context Menu – Simpler tool!
- Text editors like Atom, VS Code, PyCharm have inbuilt support for Git operations

# The Path Forward!

# What else to Do?

- Create a gist on GitHub – Small code snippets can be shared – For example, see mine at https://gist.github.com/anandjagadeesh
- Create a Wiki for your project
- Create a simple personal website or a project website – Explained in previous slides
- Learn about how to support large files at https://git-lfs.github.com/
- Learn and try advanced features in Git and GitHub

# That’s All Folks!

# About me

- Anand Jagadeesh / Anand J. / AJ
- Secretary, IEEE Computer Society Bangalore Chapter and IEEE CS Members for over 7 years
- Software Engineer 1 in PowerMax Remote Replication team of PowerMax Replication and Cloud, Dell EMC (Dell Technologies)
- A Git user – Wouldn't call myself an expert – Just a learner
- FOSS Enthusiast

# Let's Continue This Discussion

- Find me at http://www.anandj.xyz
- Send me a Feedback at: https://sl.anandj.xyz/feedback
- I have a Google Classroom for Git related discussions and chat forums and may be occasional assignments to learn.
  - Code is: 2oacxmk
  - You need an @gmail.com account to access this
  - Invite Link: https://classroom.google.com/c/NjQ0ODgwOTUzMjRa?cjc=2oacxmk
- Mail: anand.j@ieee.org | mail@anandj.xyz (preferred)
- LinkedIn: http://linkedin.anandj.xyz or https://www.linkedin.com/in/anandjagadeesh
- Some of my upcoming and past session details are at http://codemind.anandj.xyz


IEEE COMPUTER SOCIETY
Bangalore Chapter

IEEE

# Appendices

# Appendix A: Why Anand suggests SSH over HTTPS as the protocol for PUSH and FETCH?

- Well, people might have different opinions on this but here are some reasons why I suggest SSH over HTTPS as the protocol:
  - SSH is considered more secure – hence worth the initial trouble
  - I have observed it to be faster and efficient – better git aware packing while transporting objects
  - A lot of enterprises using Linux/UNIX-like environments suggest and enforces SSH over HTTPS and hence its good to learn and use SSH for Git
  - GitHub suggests SSH in GitHub Docs
- Well since people are never satisfied with the above responses, here are some links to refer and research:
  - https://docs.github.com/en/github/authenticating-to-github/using-ssh-over-the-https-port
  - https://docs.github.com/en/github/authenticating-to-github/connecting-to-github-with-ssh
  - https://docs.github.com/en/github/using-git/which-remote-url-should-i-use
  - https://git-scm.com/book/en/v2/Git-Internals-Transfer-Protocols

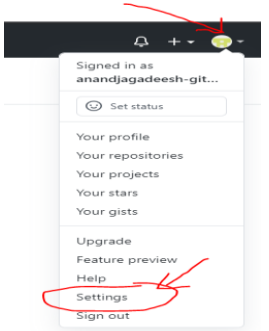
# Appendix B: How to setup SSH? The step-by-step guide

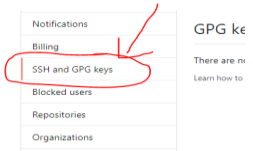
- Step 1: In your Linux terminal or in the Git Bash terminal, run

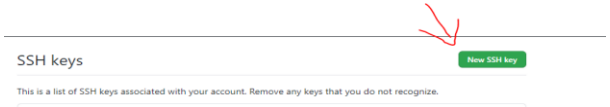
**ssh-keygen -t rsa -b 4096 -C “<your_email_address>“**

  - Leave the file name as blank – hit enter
  - Give a password or keep it blank(preferred for beginners on unimportant repositories)

- Step 2: Now, copy contents of the following file:
  - Linux: **~/.ssh/id_rsa.pub**
  - Windows: **C:\Users\<username>\.ssh\id_rsa.pub**
- Step 3: Go to GitHub.com and go to “**Settings**” from the dropdown on the profile button on top right corner of the web interface
- Step 4: Then go to “**SSH and GPG keys**” from the left menu
- Step 5: Click on the green “**New SSH key**” button
- Step 6: Give a name to the key and paste the entire contents you copied from the id_rsa.pub file in step 2
- Step 7: Save it
- You have the SSH keys generated and added to GitHub
- Now you can use SSH URL of the repo as the remote repo URL in origin or when you add a remote repo



# Appendix C: Anand's references for this workshop

- Wikipedia | https://en.wikipedia.org/wiki/Main_Page
- GitHub | https://github.com/
- Learn Git with GitKraken | https://www.gitkraken.com/resources/learn-git
- GitHub Docs | https://docs.github.com/
  - https://docs.github.com/en/github/authenticating-to-github/using-ssh-over-the-https-port
  - https://docs.github.com/en/github/authenticating-to-github/connecting-to-github-with-ssh
  - https://docs.github.com/en/github/using-git/which-remote-url-should-i-use
  - https://git-scm.com/book/en/v2/Git-Internals-Transfer-Protocols
- Pro Git – 2nd Edition (2014) – Print
- Pro Git Ebook | https://git-scm.com/book/en/v2
- Git Reference | https://git-scm.com/docs
- Git Cheat Sheets | https://github.github.com/training-kit/
- W3 Schools | https://www.w3schools.com/
- Anand's Git Classroom on Google Classroom https://classroom.google.com/c/NjQ0ODgwOTUzMjRa?cjc=2oacxmk