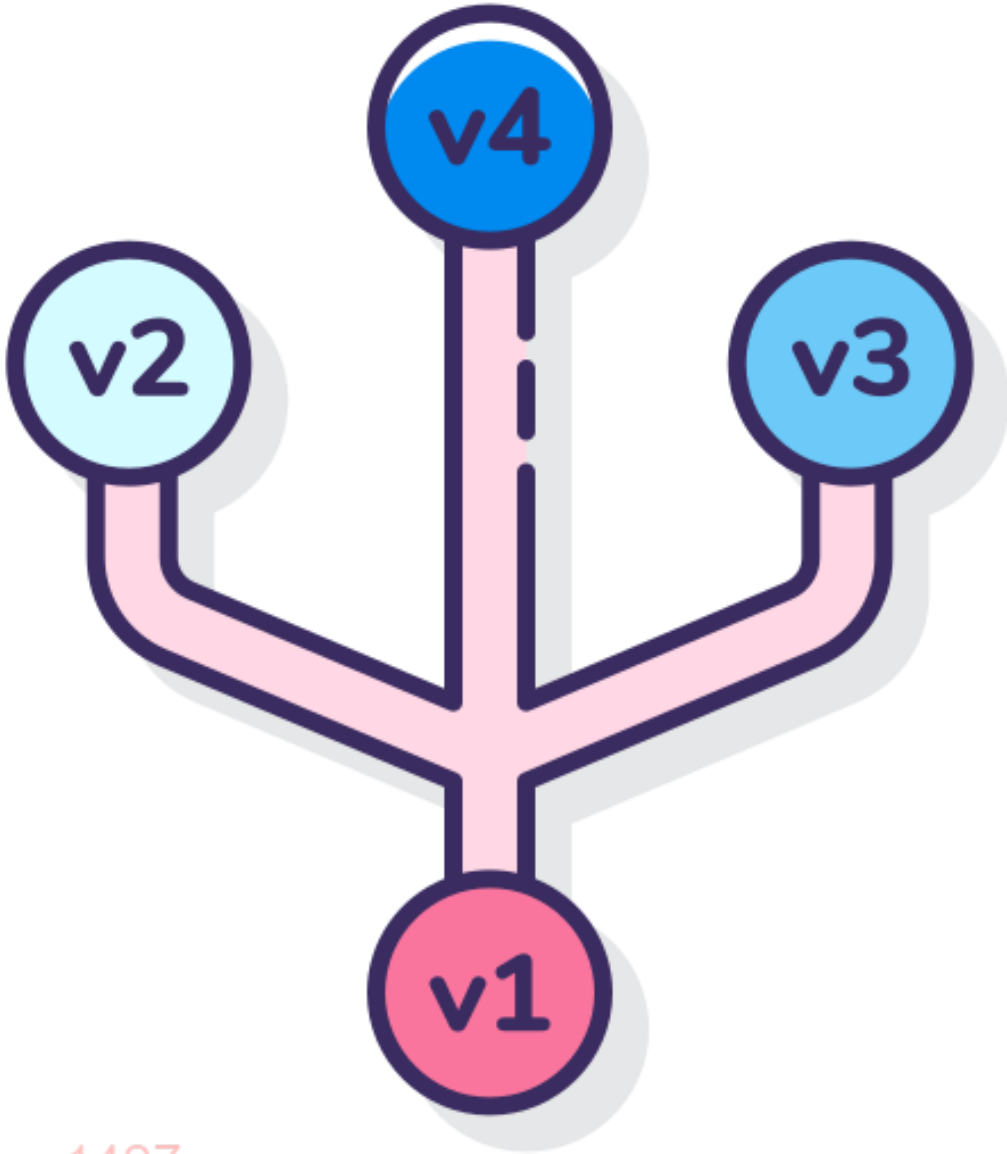
Introduction to **Git**

Distributed Version Control



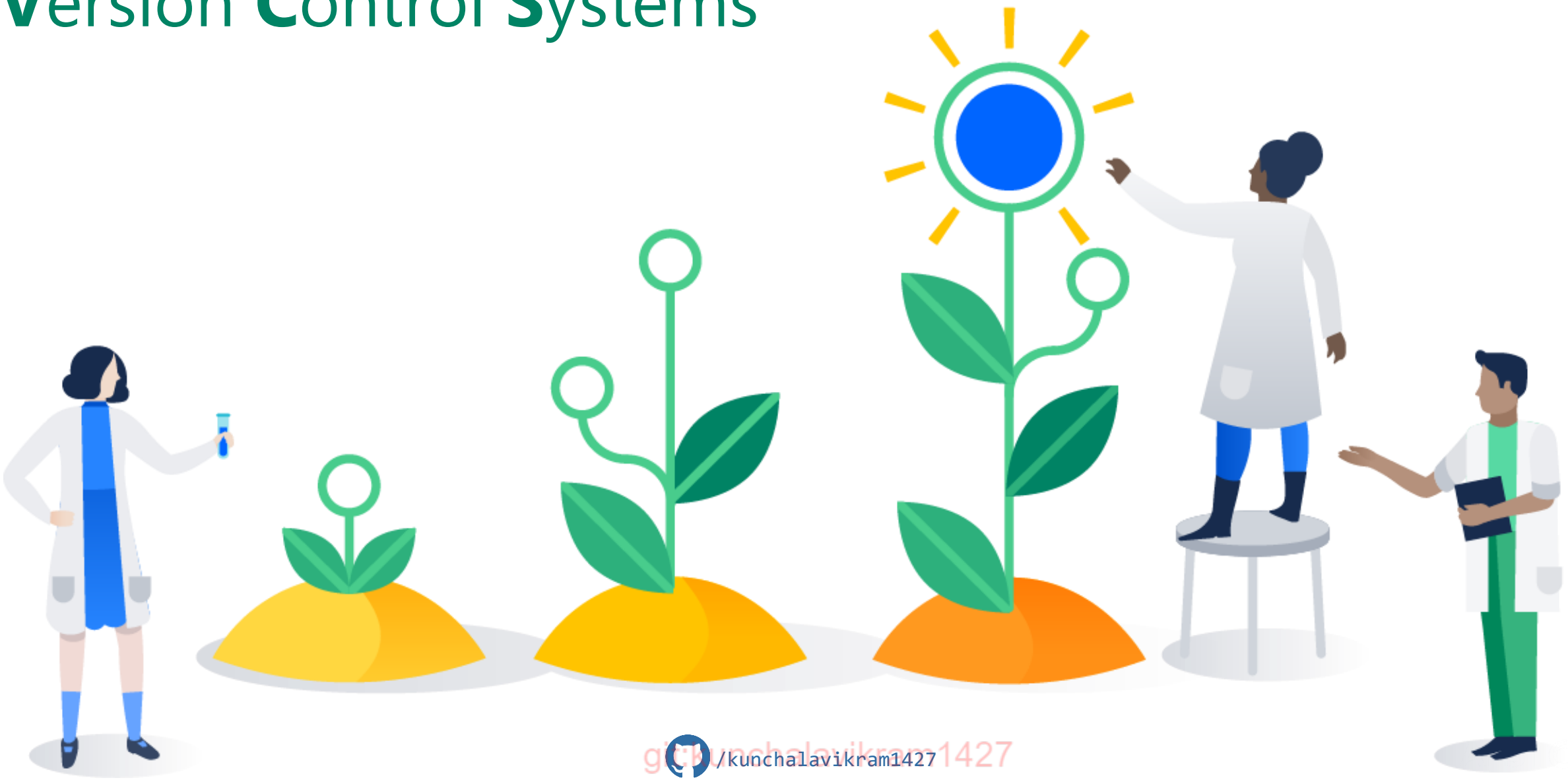
Vikram
IoT Application Dev & DevOps

git:kunchalavikram1427

Contents

- Version Control Systems
- Setup Git
- Setup Local Git Repository
- Setup Remote Repository
- Git Branches

Version **C**ontrol **S**ystems



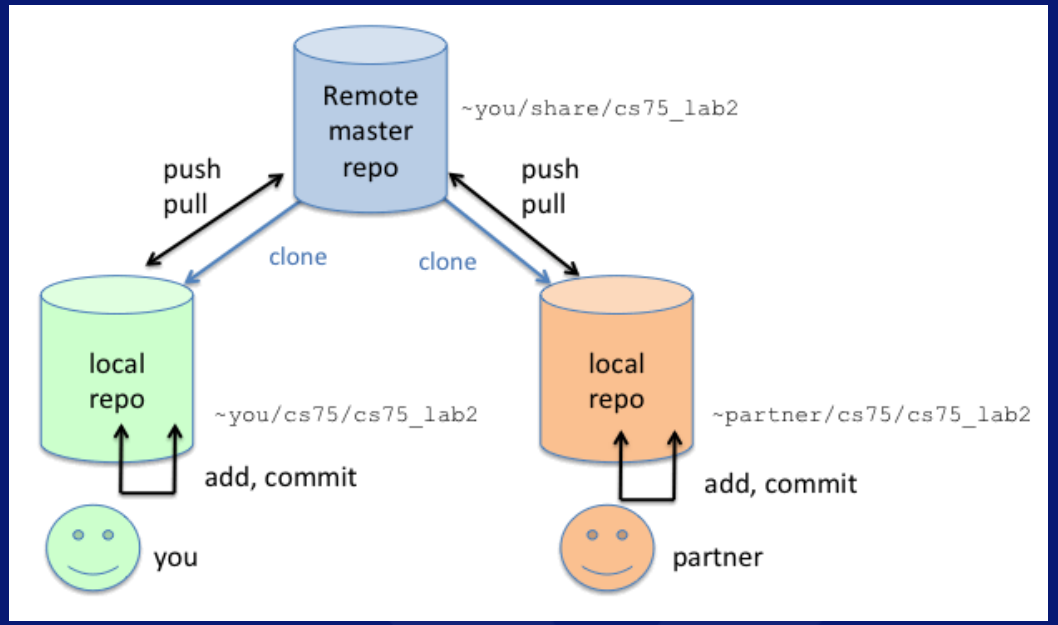
Version Control Systems(VCS)

- Version control, also known as source control, is the practice of tracking and managing changes to software code.
- It enables multiple people to simultaneously work on a single project. Each person edits his or her own copy of the files and chooses when to share those changes with the rest of the team.
- These systems are critical to ensure everyone has access to the latest code. As development gets more complex, there's a bigger need to manage multiple versions of entire products.
- Version control also enables one person you to use multiple computers to work on a project, so it is valuable even if you are working by yourself.
- Version control integrates work done simultaneously by different team members. In most cases, edits to different files or even the same file can be combined without losing any work. In rare cases, when two people make conflicting edits to the same line of a file, then the version control system requests human assistance in deciding what to do.
- Version control gives access to historical versions of your project. If you make a mistake, you can roll back to a previous version. You can reproduce and understand a bug report on a past version of your software. You can also undo specific edits without losing all the work that was done in the meanwhile. For any part of a file, you can determine when, why, and by whom it was ever edited.
- In DevOps, other than keeping track of changes, VCS also helps in developing and shipping the products faster.

Version Control Systems

Repositories and working copies

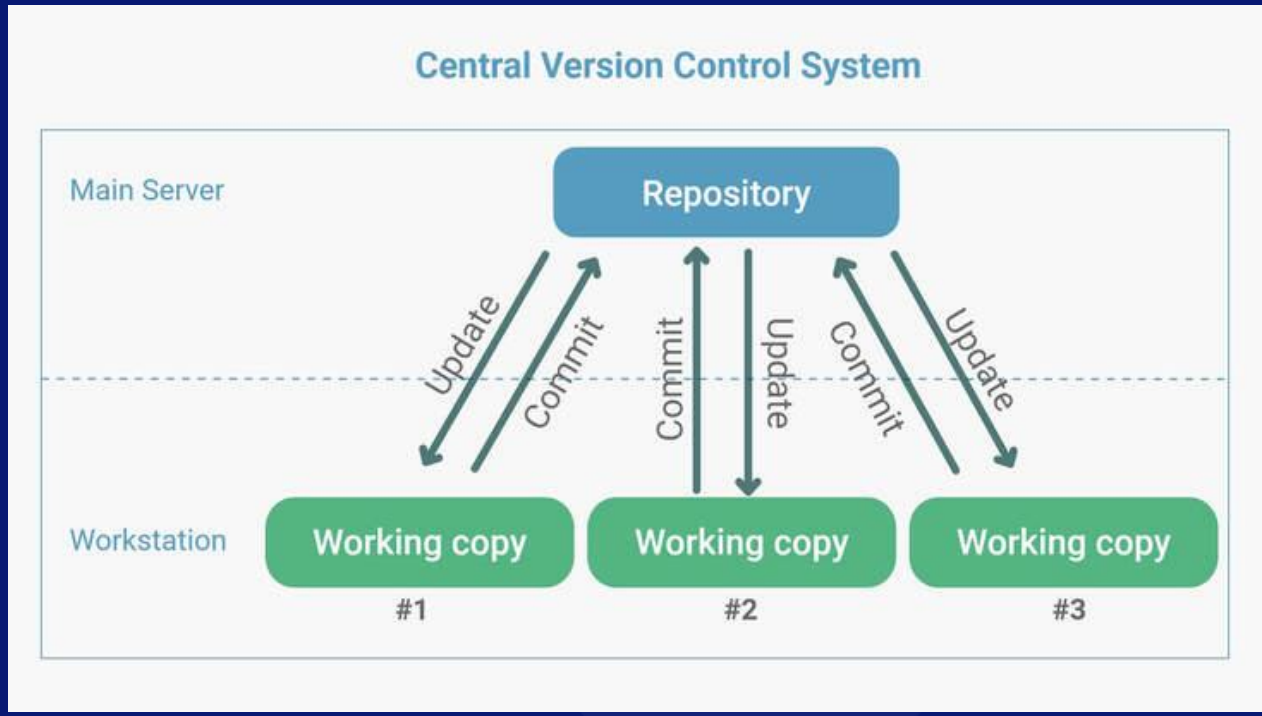
- Version control uses a remote repository and a working copy where you do your work
- Your working copy is your personal copy of all the files in the project. You make arbitrary edits to this copy, without affecting your teammates. When you are happy with your edits, you commit your changes to a repository
- A repository is a database of all the edits to, and/or historical versions (snapshots) of, your project
- It is possible for the repository to contain edits(from other developers) that have not yet been applied to your working copy
- You can update your working copy to incorporate any new edits or versions that have been added to the repository since the last time you updated(pulling the latest changes)



Central Vs Distributed VCS

Central VCS

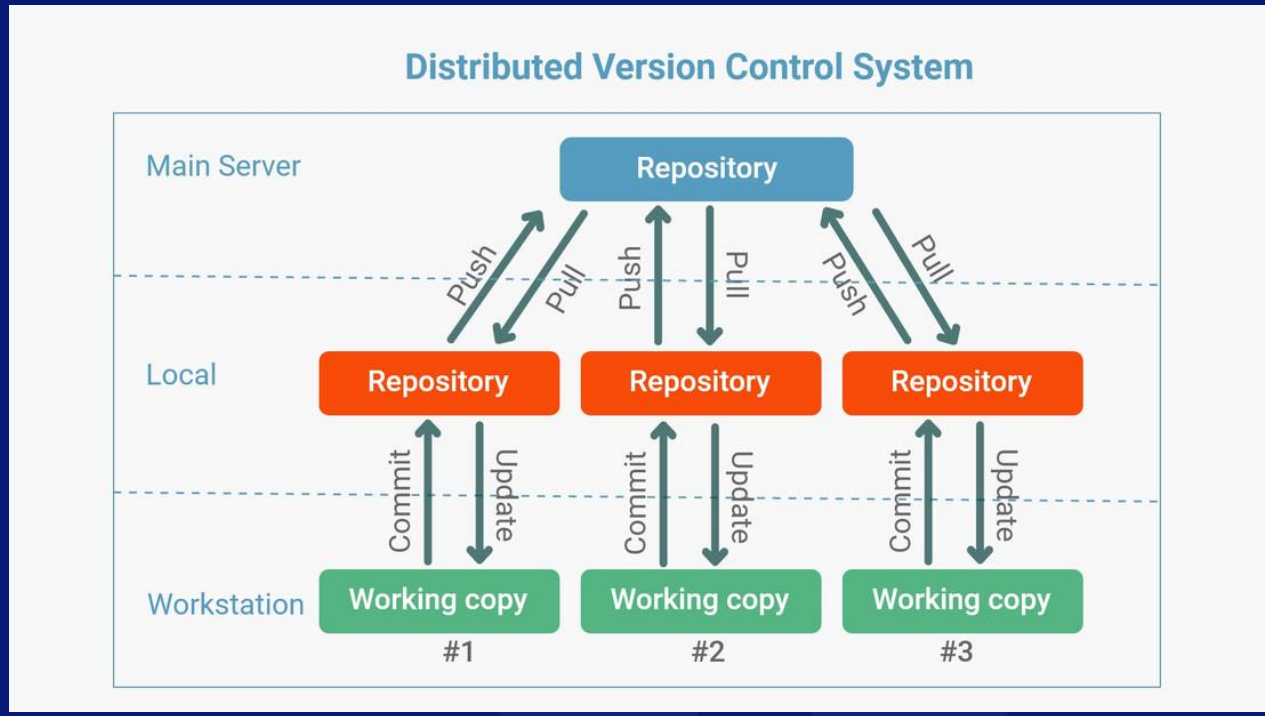
- The main difference between centralized and distributed version control is the number of repositories
- In centralized version control, there is just one repository, and in distributed version control, there are multiple repositories
- In CVCS, the central server stores all the data. Each user gets his or her own working copy, but there is just one central repository. As soon as you commit, it is possible for your co-workers to update and to see your changes
- If the central server gets crashed, there is a high chance of losing the data
- For every command, CVCS connects the central server which impacts speed of operation
- Ex: Subversion VCS



Central Vs Distributed VCS

Distributed VCS

- In distributed version control, each user gets his or her own local repository and a remote repository for all
- After you commit to the local repository, others have no access to your changes until you push your changes to the central repository
- If other users want to check your changes, they will pull the updated central repository to their local repository, and then they update in their local copy
- Even if the main server crashes, code that is in the local systems can be used to restore the data
- DVCS is fast compared to CVCS because you don't have to contact the central server for every command
- Ex: Git, Mercurial



git

Installing Git

Git

Installing Git

For Windows

- Visit https://git-scm.com/book/en/v2/Getting-Started-Installing-Git to install Git for your operating systems
- To install GUI clients, visit https://git-scm.com/downloads/guis
- Verify if Git is installed by running the command `git version` in the CMD
- Use `git help` to get the list of supported commands and `git help <sub-command>` for help on that command

```
428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git version
git version 2.27.0.windows.1
```

```
428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone             Clone a repository into a new directory
   init              Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add               Add file contents to the index
   mv                Move or rename a file, a directory, or a symlink
   restore           Restore working tree files
   rm                Remove files from the working tree and from the index
   sparse-checkout   Initialize and modify the sparse-checkout

examine the history and state (see also: git help revisions)
   bisect            Use binary search to find the commit that introduced a bug
   diff              Show changes between commits, commit and working tree, etc
   grep              Print lines matching a pattern
   log               Show commit logs
   show              Show various types of objects
   status            Show the working tree status

grow, mark and tweak your common history
   branch            List, create, or delete branches
   commit            Record changes to the repository
   merge             Join two or more development histories together
   rebase            Reapply commits on top of another base tip
   reset             Reset current HEAD to the specified state
   switch            Switch branches
   tag               Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch             Download objects and refs from another repository
   pull              Fetch from and integrate with another repository or a local branch
   push              Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git help init
```

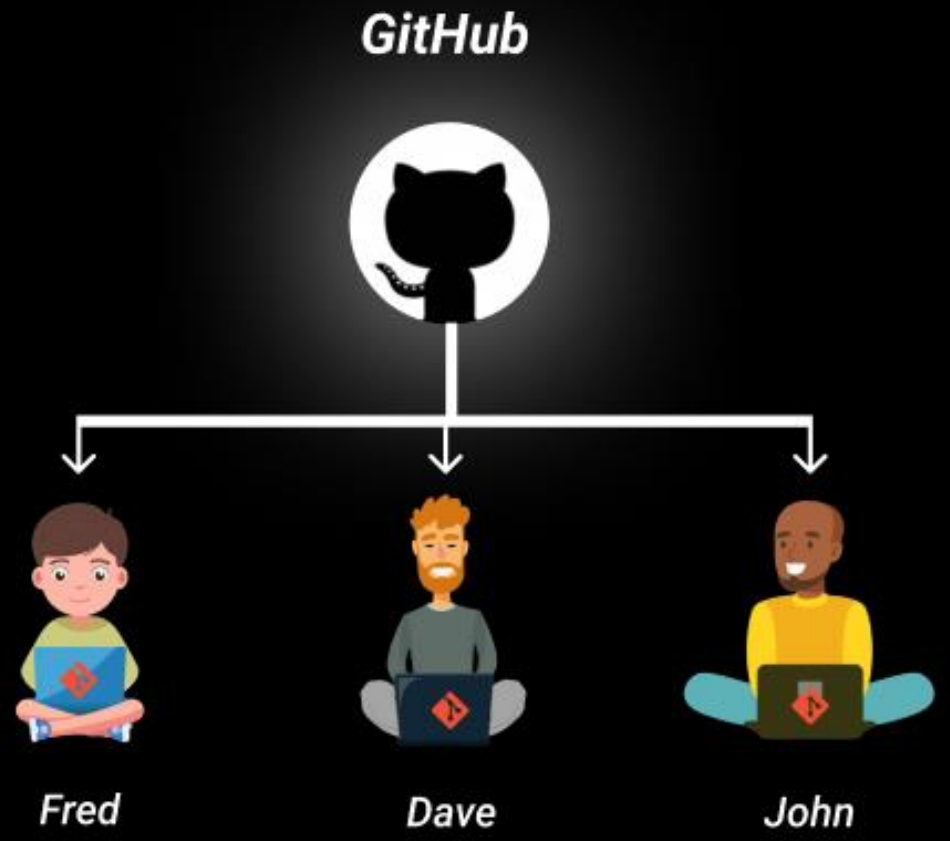
Git

What is Git?

Git is a distributed version control system source code management (SCM) system that records changes to a file or set of files over time, so that you can recall specific versions later.

It allows you to revert selected files back to a previous state, compare changes over time, see who last modified something that might be causing a problem, and more.

Git has a remote repository which is stored in a server and a local repository which is stored in the computer of each developer. This means that the code is not just stored in a central server, but the full copy of the code is present in all the developers' computers



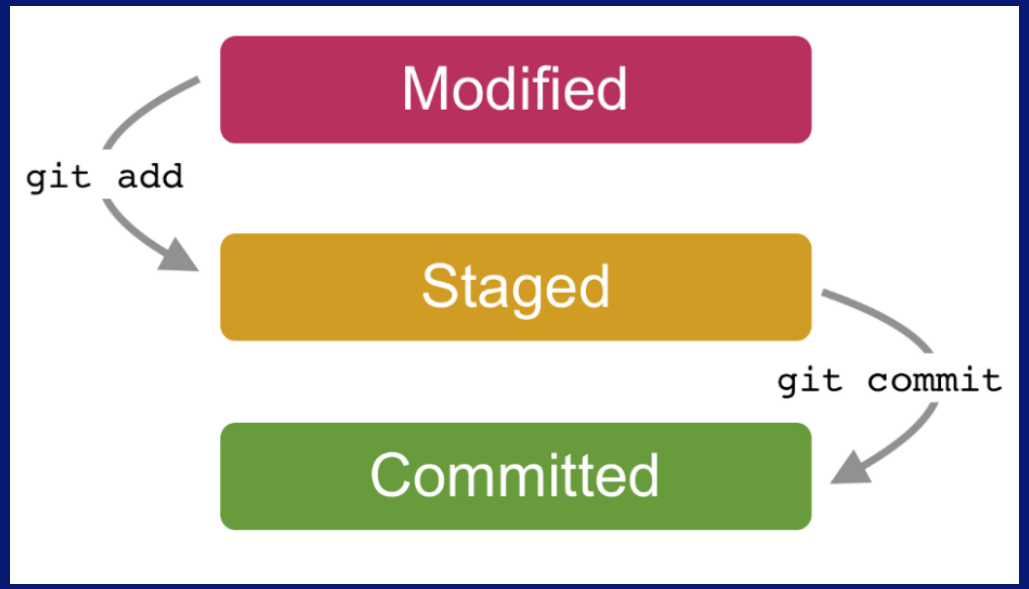
Git

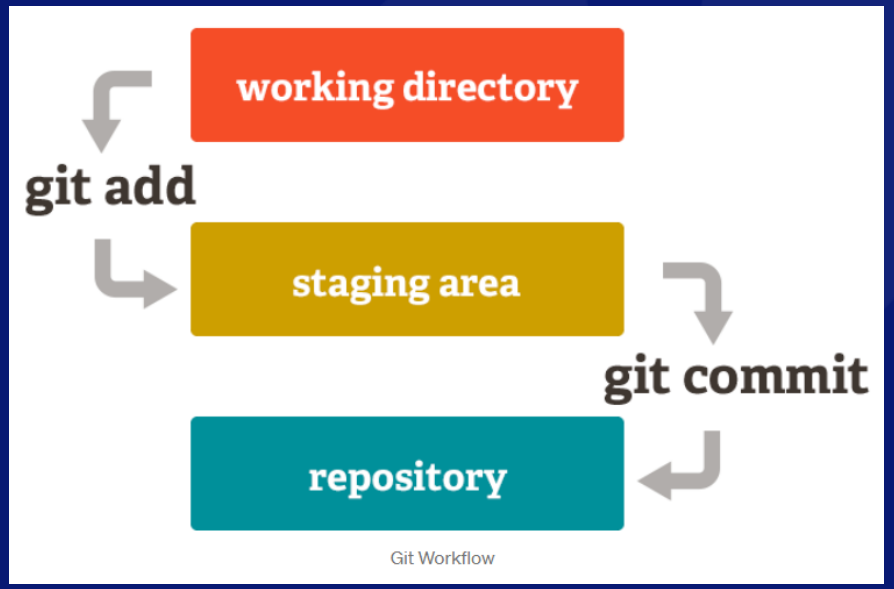
The 3 States of Git

Files in a repository go through three stages before being under version control with git

1. Modified means that you have changed the file but have not committed it to your database yet
2. Staged means that you have marked a modified file in its current version to go into your next commit snapshot
3. Committed means that the data is safely stored in your local database

- In addition to these three states, Git files live in one of three areas: the Working directory, Staging area, and the Git directory (your local repository)





Git

Create a local git repository

Create a project

- Create a new project folder in your local filesystem
- Go into this newly created project folder and add a local Git repository to the project using the following commands
  `git init`
- Now this project can be managed using Git
- Git creates a hidden folder `.git` in the project whenever you do `git init`
- The `.git` folder contains all the information that is necessary for your project in version control and all the information about commits, branches, remote repository address, etc. All of them are present in this folder. It also contains a log that stores your commit history so that you can roll back to history
- Without .git, the project is considered a local project and not a git project, that means you cannot perform any git operations

```
428991@LINL190904638 MINGW64 /d
$ mkdir git-demo

428991@LINL190904638 MINGW64 /d
$ cd git-demo

428991@LINL190904638 MINGW64 /d/git-demo
$ git init
Initialized empty Git repository in D:/git-demo/.git/

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ ls -al
total 8
drwxr-xr-x 1 428991 1049089 0 Feb 10 19:55 ./
drwxr-xr-x 1 428991 1049089 0 Feb 10 19:55 ../
drwxr-xr-x 1 428991 1049089 0 Feb 10 19:55 .git/

428991@LINL190904638 MINGW64 /d/git-demo (master)
$
```

http://gitready.com/

Git

Create a local git repository

Add files to working area

- A newly added file will always be created in the working area and is untracked by default
- Now add some files to the local repository
- Check the status of the files using `git status`
- The status shows that `file1.txt` is an `untracked file` as it is not added to the staging area or Git has no idea what to do with this file yet
- `No commits yet` indicates there are no commits to the local repository yet
- The default Git branch is `master`

```
428991@LINL190904638 MINGW64 /d/git-demo (master)
$ touch file1.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ echo "My First File" > file1.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ ls -al
drwxr-xr-x 1 428991 1049089  0 Feb 10 19:55 .git/
-rw-r--r-- 1 428991 1049089 14 Feb 10 19:57 file1.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt

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

Git

Create a local git repository

Add files to staging/index using 'add'

- Unlike many version control systems, Git has a staging area (index)
- The Staging area is there to keep track of all the files which are to be committed. Any file which is not added to the staging area will not be committed. This gives the developer control over which files need to be committed at once
- git add lets you add files to the staging area
  - `git add file1.txt`
  - `git status`
- The status shows that `file1.txt` is ready to be committed to the local repository
- In case you want to add multiple files you can use:
  - `git add file1 file2 file3`
  - `git add .` or `git add -A`

```
428991@LINL190904638 MINGW64 /d/git-demo (master)
$ ls -a
./  ../  .git/  file1.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git add file1.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt


428991@LINL190904638 MINGW64 /d/git-demo (master)
$
```

Git

Create a local git repository

Discard unstaged changes

- Before adding the files to staging, you can also discard the changes done
- Lets add some content to file1.txt and see the file status using `git status`
- The file is now in untracked state
- Git shows few hints whether to commit these changes or to discard them
- To commit simple proceed with `git add` or run `git restore <file-name>` to discard the changes
- Run `git status` again to see the status

```
428991@LINL190904638 MINGW64 /d/git-demo (master)
$ ls -a
./  ../  .git/  file1.txt  file2.txt  file3.txt  file4.txt  file5.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ cat file1.txt
My First File

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ echo "My First Modification to file1" > file1.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ cat file1.txt
My First Modification to file1

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt

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

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git restore file1.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ cat file1.txt
My First File

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Git

Create a local git repository

Add files to staging

- Add few more files as shown in the figure and add to staging area
  - `git add file2.txt file3.txt` (or)
  - `git add .`
- Once a file is in the Staging area, you can unstage the file / untrack it using the `git rm --cached <file_name>` command

```
428991@LINL190904638 MINGW64 /d/git-demo (master)
$ touch file2.txt file3.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ echo "My Second File" > file2.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ echo "My Third File" > file3.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt

Untracked files:
  (use "git add <file>..." to include in what will
be committed)
        file2.txt
        file3.txt


428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git add .

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt
        new file:   file2.txt
        new file:   file3.txt
```

Git

Create a local git repository

Unstage files

- `git reset <filename>, git reset HEAD <filename>, git rm --cached <filename>, git restore --staged <filename>` all does the same work

```
428991@LINL190904638 MINGW64 /d/git-demo (master)
$ echo "My First Modification to file1" > file1.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt

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

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git add .

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   file1.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git reset HEAD
Unstaged changes after reset:
M       file1.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt
```

Git

Git Config

- The 'Git config' is a great method to configure your choice for the Git installation. Using this command, you can describe the repository behaviour, preferences, and user information
- The global config resides at `~/.gitconfig` in the user's home directory, while the repository specific config at `.git/config` inside the project
- Use `git config --help` for more info
- Before we commit any changes to the repo, we need to set the user name and email address for the git user so as to make Git aware of who has made the commits
- Run below commands to set the configuration
- `git config --global user.name "your name goes here"`
- `git config --global user.email "your email goes here"`

```
428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git config --global user.name "vikram"

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git config --global user.email "vikram.k@gmail.com"

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git config --global --list
http.sslverify=false
filter.lfs.required=true
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
user.name=vikram
user.email=vikram.k@gmail.com
core.editor="C:/Program Files (x86)/GitExtensions/GitExtensions.exe" fileeditor


428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git config --list
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
http.sslbackend=openssl
diff.astextplain.textconv=astextplain
core.autocrlf=true
core.fscache=true
core.symlinks=false
core.editor="C:\\Program Files\\Notepad++\\notepad++.exe" -multiInst -notabbar -
nosession -noPlugin
credential.helper=manager
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
pull.rebase=false
http.sslverify=false
filter.lfs.required=true
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
user.password=Virender418@
user.email=vikram.k@gmail.com
core.editor="C:/Program Files (x86)/GitExtensions/GitExtensions.exe" fileeditor
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
```

git:kunchalavikram1427

Git

Create a local git repository

Committing the code

- Committing is the process in which the code is added to the local repository
- Each commit has an associated commit message, which is a description explaining why a particular change was made. Commit messages capture the history of your changes, so other contributors can understand what you've done and why
- To give commit message, use -m flag
  - `git commit -m "Initial Commit"`
- If you don't use -m, Git will bring up an editor for you to write the commit message
- Each time you commit changes to the repo, Git creates a new SHA(20 byte number) that describes that state

```
428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git commit -m "Initial Commit"
[master (root-commit) 35b408c] Initial Commit
 Committer: vikram <vikram.k@gmail.com>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 3 files changed, 3 insertions(+)
 create mode 100644 file1.txt
 create mode 100644 file2.txt
 create mode 100644 file3.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$
```

Git

Create a local git repository

View commit history

- `git log` command lists the commits made in that repository in reverse chronological order
- From the logs you can see the commit SHA ID, Author, Date and Commit message
- The **HEAD** points out the last commit in the current checkout branch. When you switch branches with `git checkout`, the HEAD is transferred to the new branch

```
428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git log
commit 53b694153e68bbed93e9a6cfc0c94d97c759b828 (HEAD -> master)
Author: 428991 <vikram.k@gmail.com>
Date:   Wed Feb 10 23:50:22 2021 +0530

    Initial Commit

428991@LINL190904638 MINGW64 /d/git-demo (master)
$
```

Git

Create a local git repository

Add few more files to the staging

- `git log`
- `git show head` is used to check the status of the Head

```
428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git log
commit 2fedc59b1199dfb2532903c7e378112f79dc714e (HEAD -> master)
Author: 428991
Date:   Thu Feb 11 08:16:27 2021 +0530

    Second Commit, Added file4.txt and file5.txt

commit 53b694153e68bbed93e9a6cfc0c94d97c759b828
Author: 428991
Date:   Wed Feb 10 23:50:22 2021 +0530

    Initial Commit

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git show head
commit 2fedc59b1199dfb2532903c7e378112f79dc714e (HEAD -> master)
Author: 428991 <vikram.babu-kunchala@alstomgroup.com>
Date:   Thu Feb 11 08:16:27 2021 +0530

    Second Commit, Added file4.txt and file5.txt

diff --git a/file4.txt b/file4.txt
new file mode 100644
index 0000000..6cafdcb
--- /dev/null
+++ b/file4.txt
@@ -0,0 +1 @@
+My Fourth File
diff --git a/file5.txt b/file5.txt
new file mode 100644
index 0000000..e5d949e
--- /dev/null
+++ b/file5.txt
@@ -0,0 +1 @@
+My Fifth File
```

```
428991@LINL190904638 MINGW64 /d/git-demo (master)
$ touch file4.txt file5.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ echo "My Fourth File" > file4.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ echo "My Fifth File" > file5.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file4.txt
        file5.txt

nothing added to commit but untracked files present (use "git add" to
track)

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git add .

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   file4.txt
        new file:   file5.txt


428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git commit -m "Second Commit, Added file4.txt and file5.txt"
[master 2fedc59] Second Commit, Added file4.txt and file5.txt
 Committer: 428991 <vikram.babu-kunchala@alstomgroup.com>
Your name and email
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 2 files changed, 2 insertions(+)
 create mode 100644 file4.txt
 create mode 100644 file5.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Git

Create a local git repository

View commit history

- `git log --patch -2` shows the difference (the patch output) introduced in each commit. You can also limit the number of log entries displayed, such as using -2 to show only the last two entries
- `git log --stat` shows abbreviated stats for each commit
- `git log --pretty=oneline` (or) `git log --oneline` --pretty changes the log output to formats other than the default. --oneline value for this option prints each commit on a single line

```
428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git log --stat
commit 2fedc59b1199dfb2532903c7e378112f79dc714e (HEAD -> master)
Author: 428991 <vikram.k@gmail.com>
Date:   Thu Feb 11 08:16:27 2021 +0530

    Second Commit, Added file4.txt and file5.txt

 file4.txt | 1 +
 file5.txt | 1 +
 2 files changed, 2 insertions(+)

commit 53b694153e68bbed93e9a6cfc0c94d97c759b828
Author: 428991 <vikram.k@gmail.com>
Date:   Wed Feb 10 23:50:22 2021 +0530

    Initial Commit

 file1.txt | 1 +
 file2.txt | 1 +
 file3.txt | 1 +
 3 files changed, 3 insertions(+)

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git log --pretty=oneline
2fedc59b1199dfb2532903c7e378112f79dc714e (HEAD -> master) Second Commit,
Added file4.txt and file5.txt
53b694153e68bbed93e9a6cfc0c94d97c759b828 Initial Commit
```

Git

Git ignore (.gitignore)

- Git sees every file in your working copy as one of three things: tracked - a file which has been previously staged or committed; untracked - a file which has not been staged or committed; or ignored - a file which Git has been explicitly told to ignore
- Git is meant to store only source files and never the generated files or the personal config files with confidential information
- Any generated code/binaries should not be committed due to following reasons:
  - The generated files can be recreated at any time on the client side and may need to be created in a different form, so It is a waste of time and space to store them in git
  - The generated files are often large than the original source files themselves. Putting them in the repo means that everyone needs to download and store those generated files, even if they're not using them. This increased the clone times
  - Git usually stores only changes made to the file in the subsequent commits by using diff algorithm. Binary files don't really have good diff tools, so Git will frequently just need to store the entire file each time it is committed
- Any file added to .gitignore file in the root project directory will not show up as untracked file while doing git status and so can be ignored. Common files to be ignored include:
  - dependency caches, such as the contents of /node_modules or /packages
  - compiled code, such as .o, .pyc, and .class files
  - build output directories, such as /bin, /out, or /target
  - files generated at runtime, such as .log, .lock, or .tmp
  - hidden system files, such as .DS_Store or Thumbs.db
  - personal IDE config files, such as .idea/workspace.xml

Git Ignore patterns:
https://www.atlassian.com/git/tutorials/saving-changes/gitignore
https://github.com/github/gitignore

Git

Git ignore (.gitignore)

- Let's create a file called TODO.txt which has todo list for personal reference
- Add some content to this file and check the status
- Git shows the file as untracked. But these are personal preferences and should not be committed or bring under Git's radar
- We can add this file to `.gitignore` file to instruct git to ignore the changes done to TODO file and not to track
- We should commit `.gitignore` file so others can use it and updated it with their own to be ignored content like files generated code, log files, cache files and IDE specific files

```
428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git log --oneline
755a2fc (HEAD -> master) Added .gitignore
2fedc59 Second Commit, Added file4.txt and file5.txt
53b6941 Initial Commit
```

```
428991@LINL190904638 MINGW64 /d/git-demo (master)
$ echo "[TODO]: Change the login button layout" > TODO.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        TODO.txt

nothing added to commit but untracked files present (use "git add" to track)

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ echo "TODO.txt" >> .gitignore

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ cat .gitignore
TODO.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ ls -a
./  ../  .git/  .gitignore  file1.txt  file2.txt  file3.txt  file4.txt  file5.txt
TODO.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

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

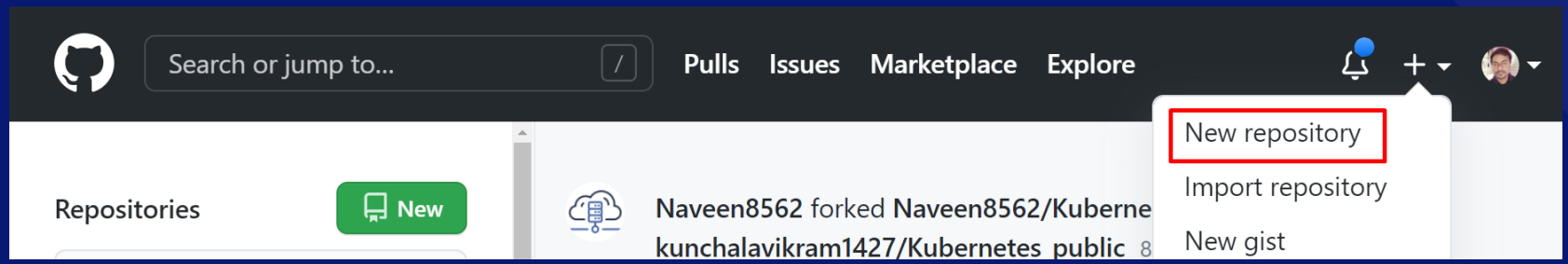
Git

Create a Remote Repo

Remote Repo in Github

- Files/Project which you have created earlier stays within your local repository
- If you want to share these files with other developers or team members to collaborate, you need to push the local repo to a remote repository like GitHub
- After you have a remote repository set up, you upload (push) your files and revision history to it. After someone else makes changes to a remote repo, you can download (pull) their changes into your local repo

- Go to https://github.com/ and create a New Repository

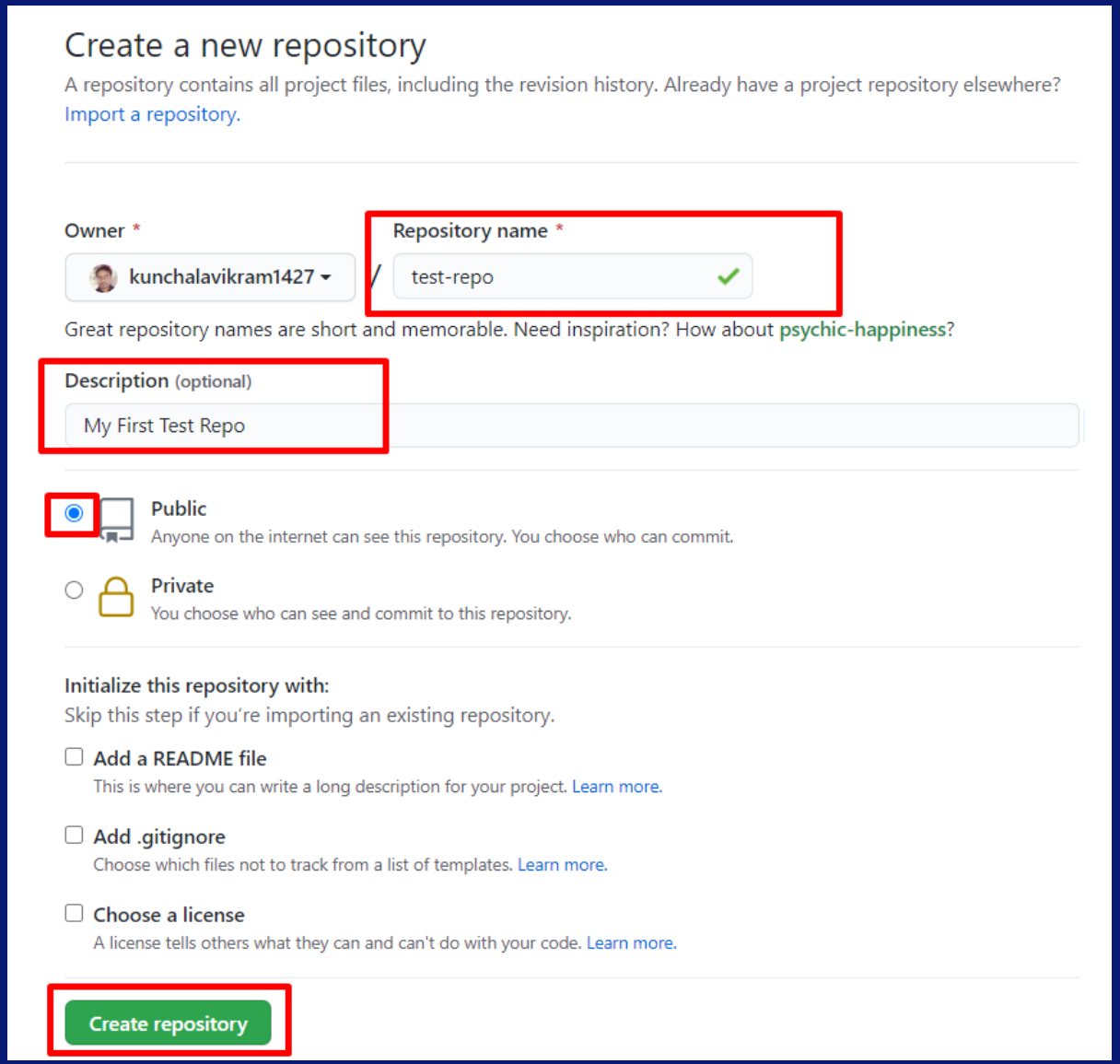


Git

Create a Remote Repo

Remote Repo in Github

- Give a Repository name, Description and Make the repo public
- Click on Create Repository to create it



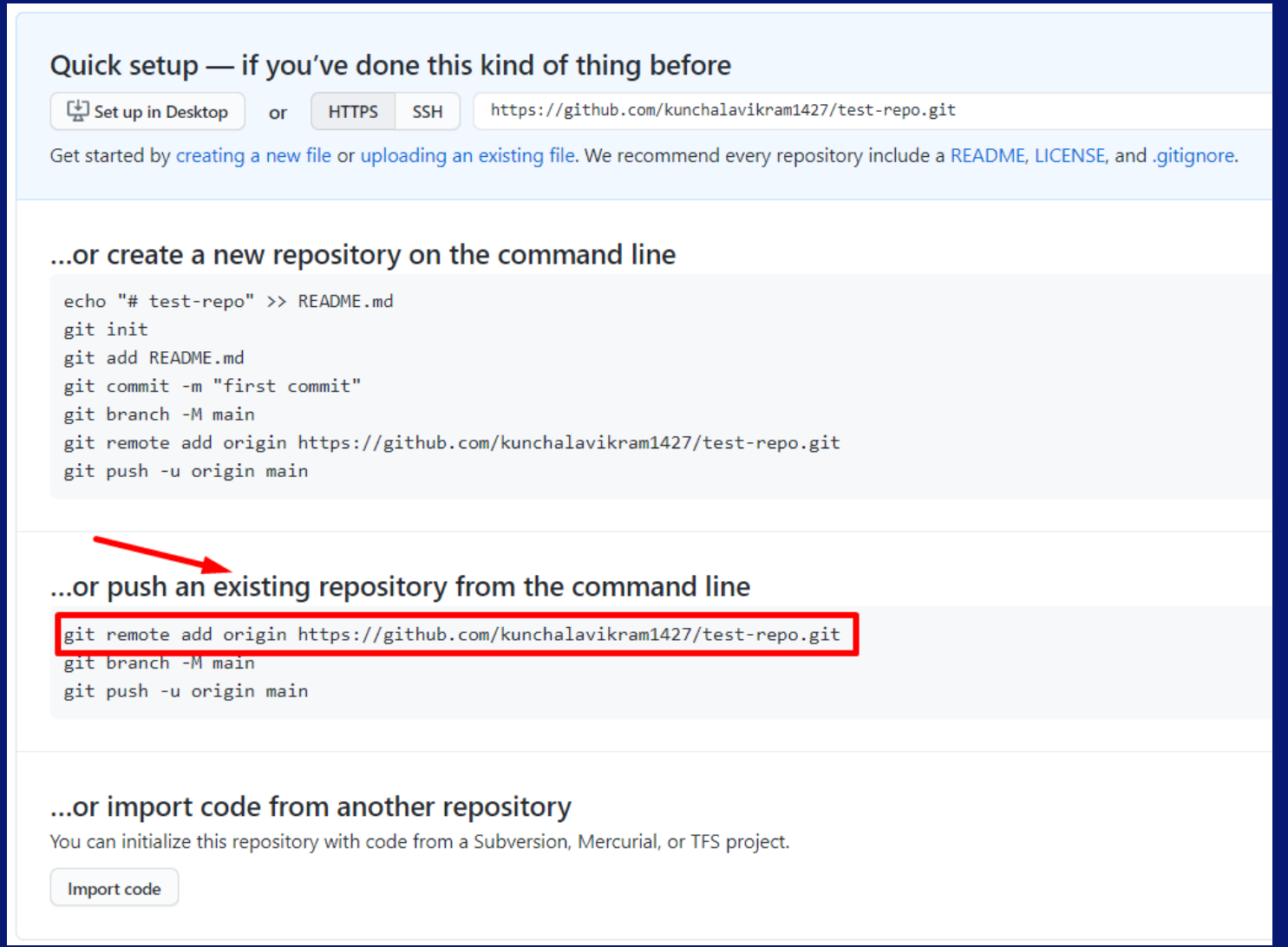
git:kunchalavikram1427

Git

Create a Remote Repo

Remote Repo in Github

- Once created, GitHub will prompt to create a new repo or to add a repo we have created locally
- In our case, since we've already created a local repository, we can push it directly
- Git will also show us the instructions to be followed to push our local repo.
- In this case we need to push to the URL that is shown



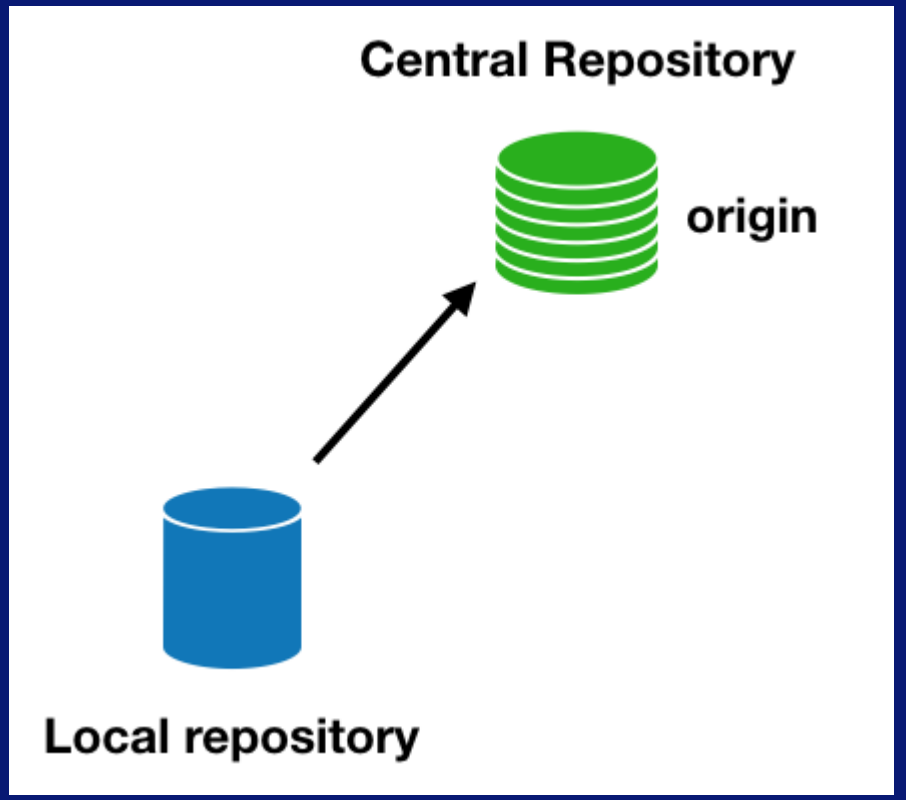
git:kunchalavikram1427

Git

Create a Remote Repo

What Is Origin?

- `Origin` is an alias to the remote repository
- We assign an alias so we won't have to write out the URL of the remote repo every time we want to work with the remote repository
- While `origin` is the alias that most people prefer to use, it is not a standard and we can use our own naming conventions



Git

Remote Repo

Adding the remote repo

- Add the remote repo by running
  - `git remote add origin https://github.com/<your-user-account>/test-repo.git`
- `remote` command adds a remote repository to the current repository
- Check the remote URLs and their aliases
  - `git remote -v`

- Remove the remote repo by running
  - `git remote rm origin`

```
428991@LINL190904638 MINGW64 /d/git-demo (dev)
$ git remote add origin https://github.com/kunchalavikram1427/test-repo.git

428991@LINL190904638 MINGW64 /d/git-demo (dev)
$ git remote -v
origin  https://github.com/kunchalavikram1427/test-repo.git (fetch)
origin  https://github.com/kunchalavikram1427/test-repo.git (push)

428991@LINL190904638 MINGW64 /d/git-demo (dev)
$ git config --list
.
.
.
user.name=vikram
remote.origin.url=https://github.com/kunchalavikram1427/test-repo.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
```

Git

Remote Repo

Push the files to the origin

- `git push` pushes our local repo changes to the added remote repo. **origin** represents the remote repo and **master** is the branch you want to push to. Git will prompt for your git credentials during push
  - `git push -u origin master`

- We can omit origin and -u master parameters from the git push command with the following two Git configuration changes:
  $ git config remote.pushdefault origin
  $ git config push.default current
- The first setting saves you from typing origin every time. And with the second setting, Git assumes that the remote branch on the GitHub side will have the same name as your local branch.

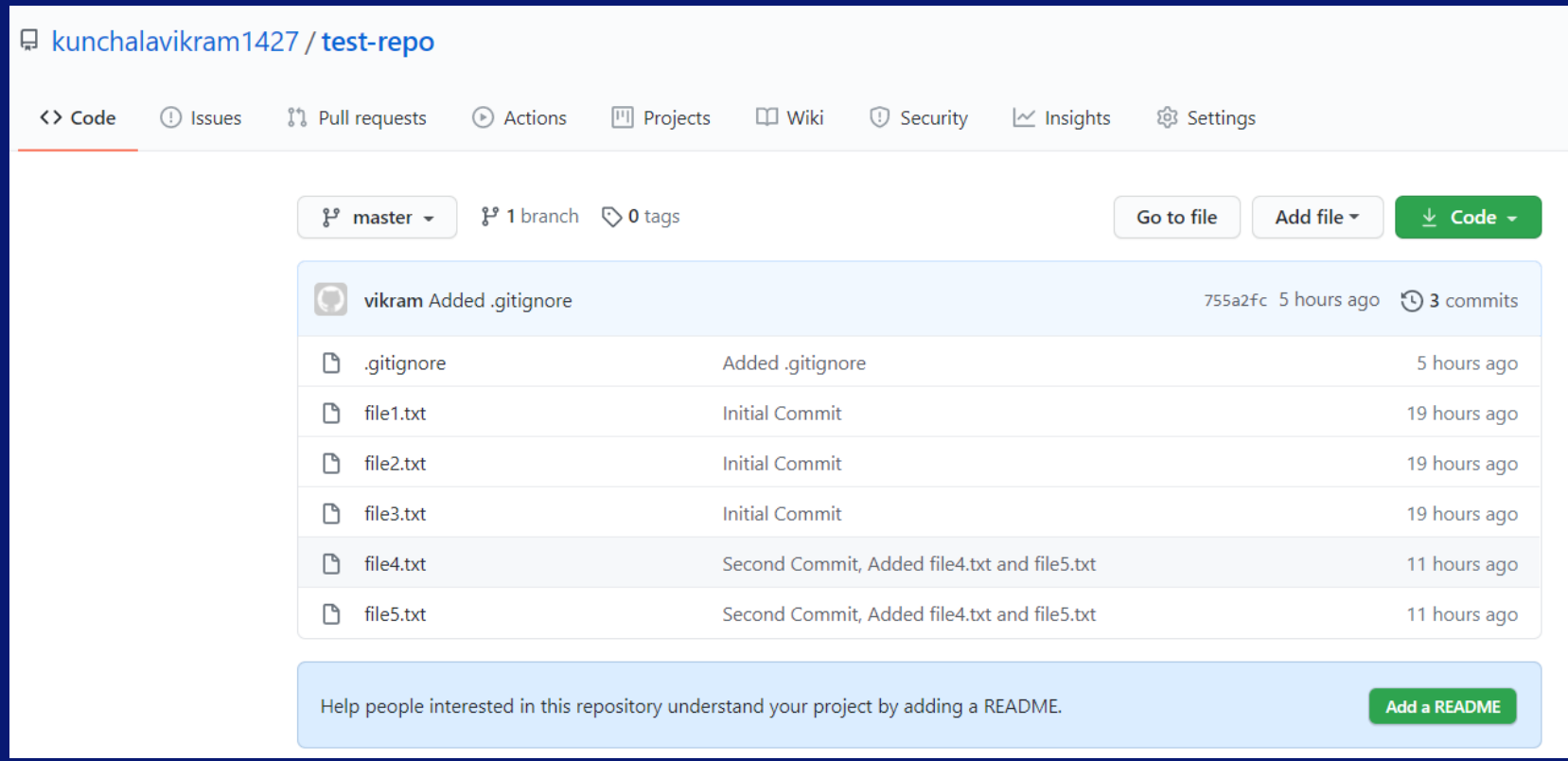
```
428991@LINL190904638 MINGW64 /d/git-demo (dev)
$ git push -u origin master
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (12/12), 854 bytes | 427.00 KiB/s, done.
Total 12 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/kunchalavikram1427/test-repo.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

428991@LINL190904638 MINGW64 /d/git-demo (dev)
$
```

Git

Remote Repo

Push the files to the origin

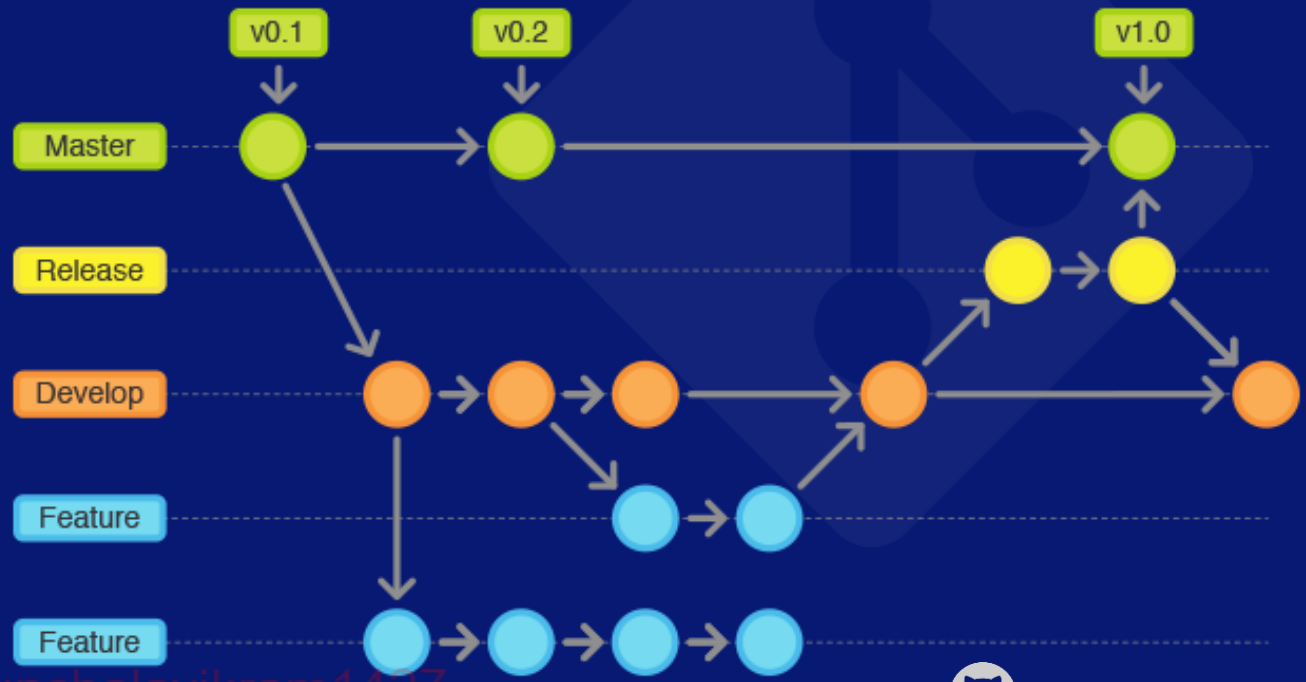
- Go to your GitHub account and see the changes



Git

Branches

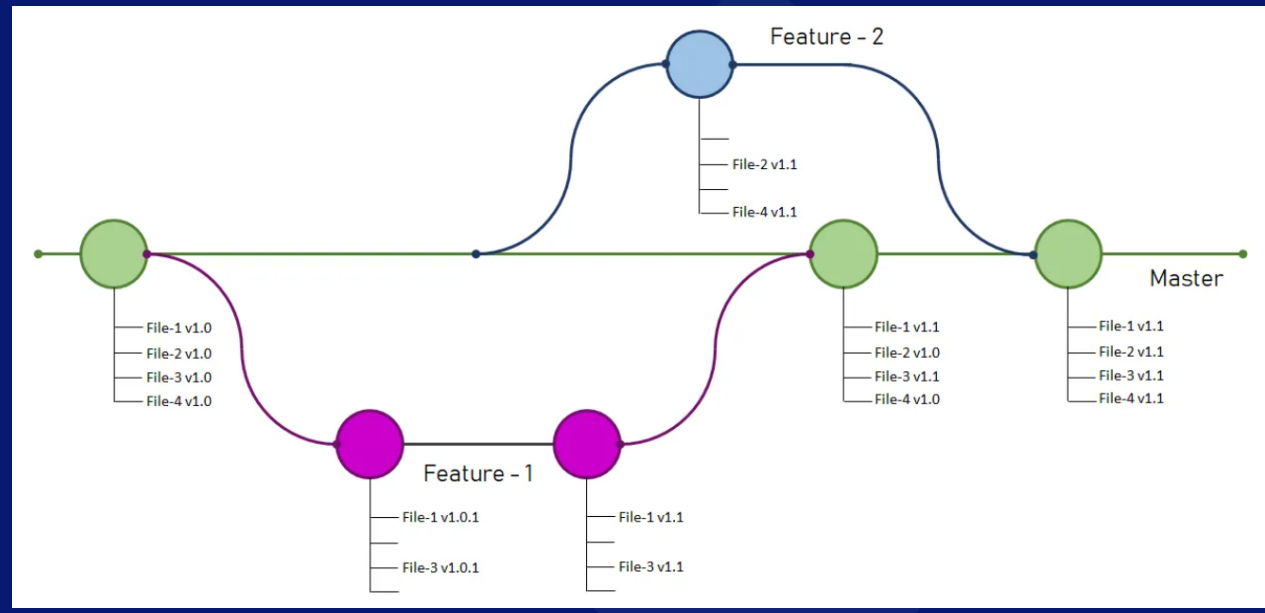
- GIT Branch is about maintaining the separate line of development. The default branch is ***Master***
- Git lets you branch out from the original code base. This lets you more easily work with other developers, and gives you a lot of flexibility in your workflow
- Let's say you need to work on a new feature for a website. You create a new branch and start working. You haven't finished your new feature, but you get a request to make a rush change that needs to go live on the site today. You switch back to the master branch, make the change, and push it live. Then you can switch back to your new feature branch and finish your work. When you're done, you merge the new feature branch into the master branch, and both the new feature and rush change are kept!



Git

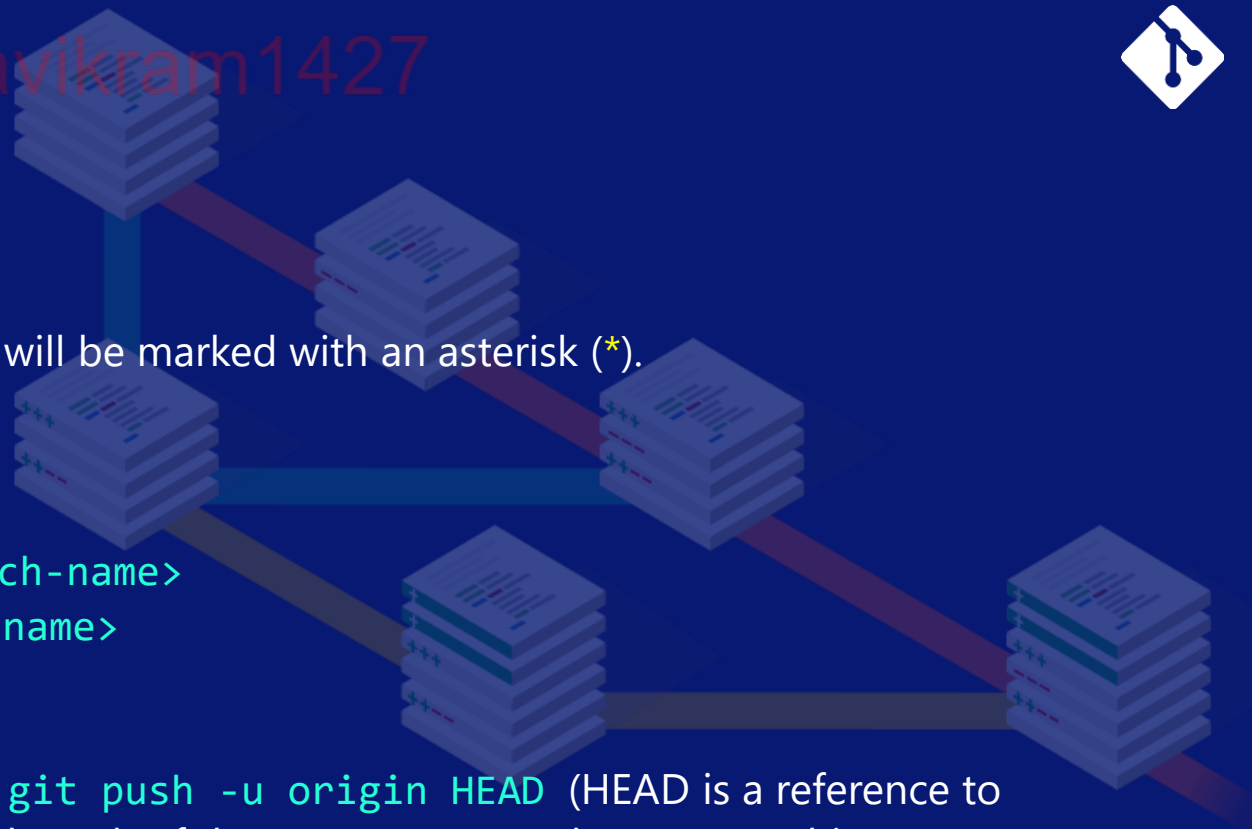
Branches

- Basically, A branch is the isolated and independent line of developing the feature
- Assume the middle line as the master branch where the code is stable, working and updated
- Then, assume a developer-1 is assigned to develop Feature – 1. So, Developer-1 is cutting a branch from the master branch which is indicated in the diagram as in Magenta colour
- Similarly Developer-2 is assigned to develop Feature-2 and he is cutting a branch from the master branch which is indicated in Blue colour
- When both of them are done with their changes they merge their branches to the master and these feature branches can be safely deleted
- Let's assume the codebase is the set of four files called File-1, File-2, File-3, and File-4. So, when Developer-1 is taking the branch from master to develop feature-1 (assume File-1 and File-3 are going to be changed for the development). So, the developer can safely commit his changes on both files into the master branch. Vice versa, When Developer-2 is taking the branch from master to develop feature-2 (assume File-2 and File-4 are going to be changed for the development).



Git

Branch commands

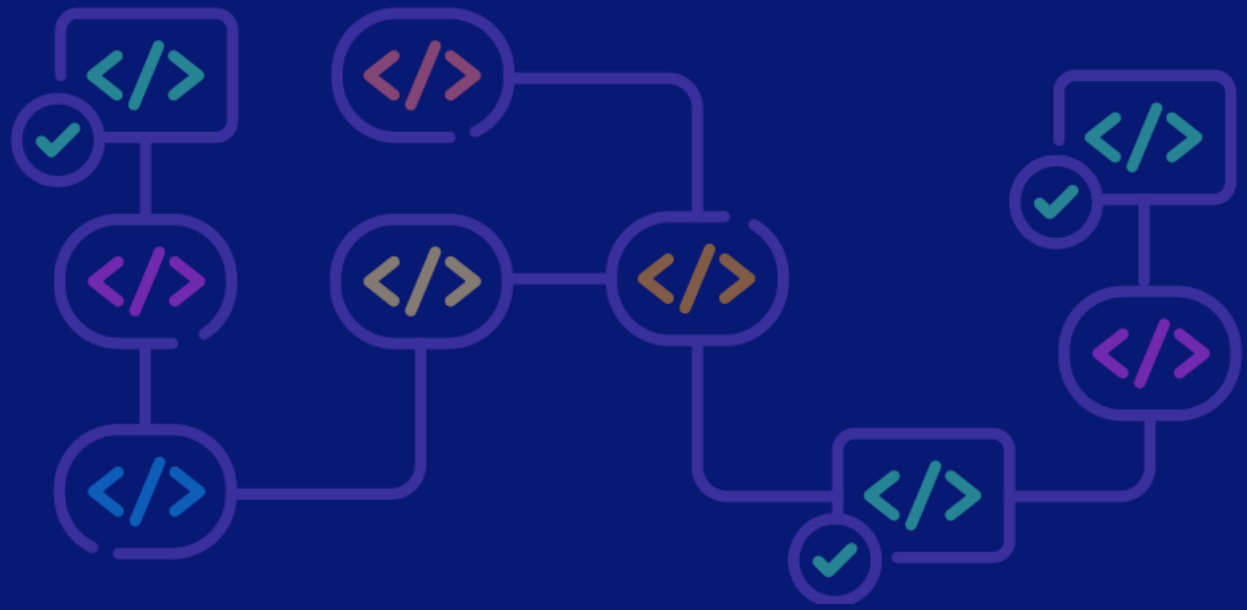


- Check What Branch You're On: `git status`
- To see local branches: `git branch` (The current local branch will be marked with an asterisk (*).
- To see remote branches: `git branch -r`
- To see all local and remote branches: `git branch -a`
- Create a New Branch: `git branch <branch-name>`
- Switch to a Branch In Your Local Repo: `git checkout <branch-name>`
- Create a branch and checkout: `git checkout -b <branch-name>`
- Rename a Branch: `git branch -m <branch-name>`
- To get a list of all branches from the remote: `git pull`
- Push to a Branch: `git push -u origin <branch-name>` or `git push -u origin HEAD` (HEAD is a reference to the top of the current branch, so it's an easy way to push to a branch of the same name on the remote. This saves you from having to type out the exact name of the branch!
- If your local branch already exists on the remote: `git push`
- You'll want to make sure your working tree is clean and see what branch you're on: `git status`
- `git merge <branch-name>` (If you want this branch to merge to master, you should already be in master branch before merging. If not run `git checkout master`)
- To delete a remote branch : `git push origin --delete <branch-name>`
- To delete a local branch: `git branch -d <branch-name>` (or) `git branch -D <branch-name>` (the -d option only deletes the branch if it has already been merged. The -D option is a shortcut for --delete --force, which deletes the branch irrespective of its merged status)

Git

Branch commands

- Delete a remote branch: `git push origin --delete :<branch-name>` or `git push origin :<branch_name>`
- Rename a local branch by being in the same branch: `git branch -m <new-branch-name >`
- Rename a local branch by being in another branch: `git branch -m <old-branch-name > <new-branch-name >`
- If you are in the branch which needs to be renamed in remote, then pass the following command with upstream argument -u: `git push origin -u new-name`

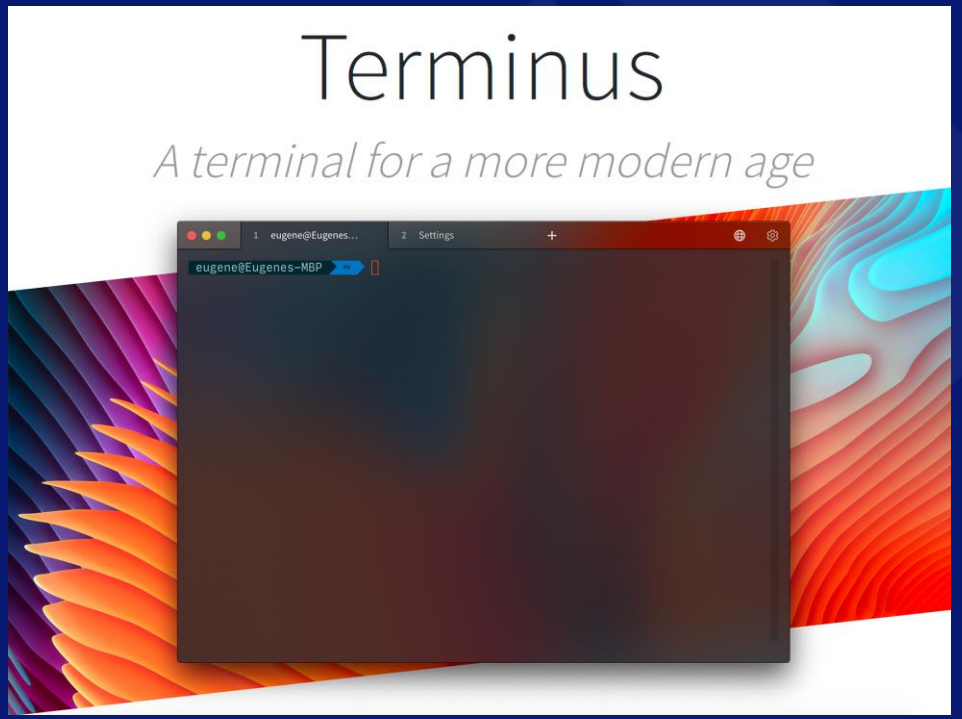


Git

CLI

- Git can be used using a command line (terminal), or a desktop app that has a GUI (graphical user interface) such as SourceTree shown next
- I prefer using terminus https://github.com/Eugeny/terminus which is a CLI tool. It directly shows the current git branch as seen below and has many features like Serial, SSH protocols
- Download it from here https://github.com/Eugeny/terminus/releases/tag/v1.0.132

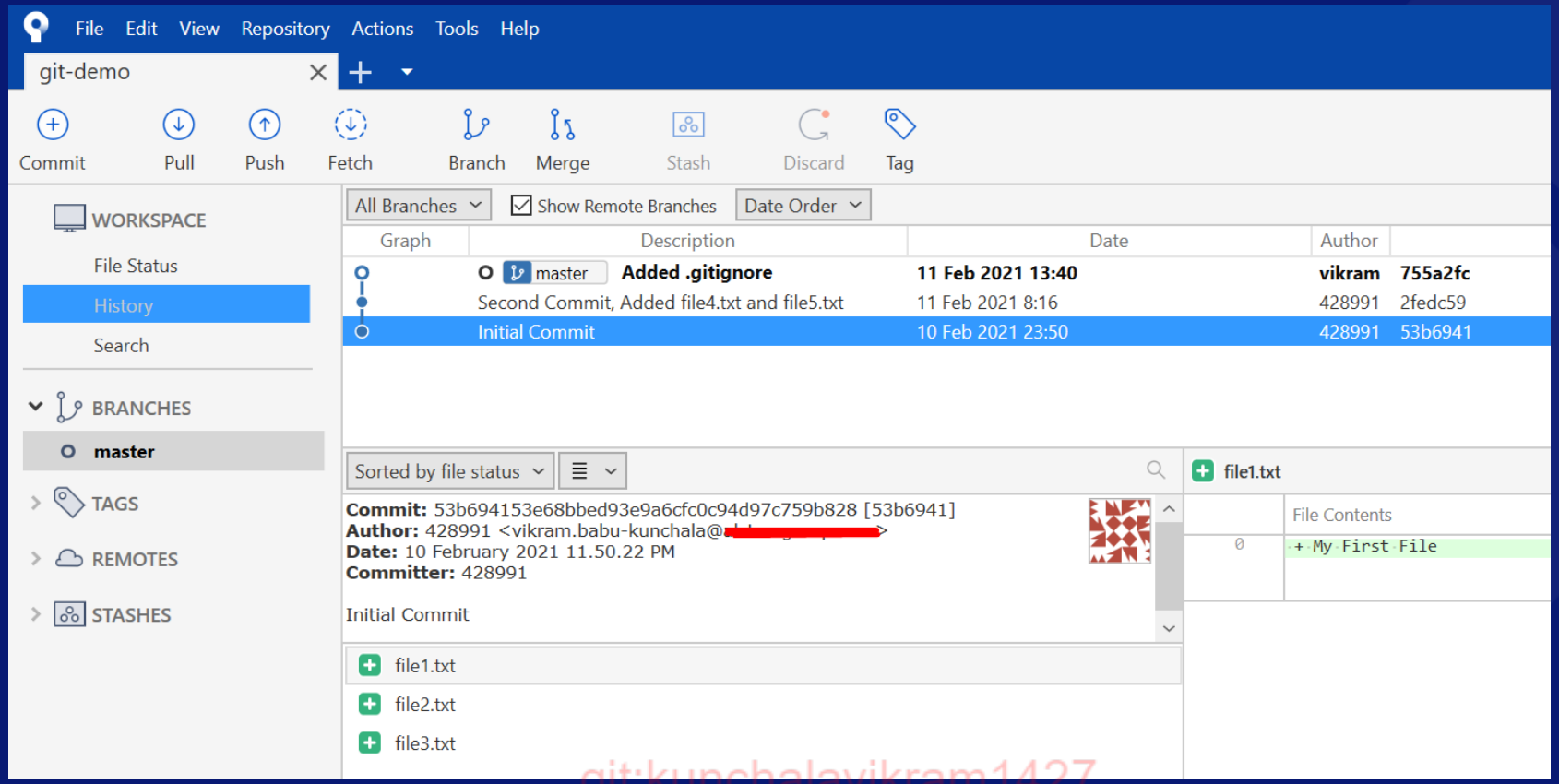
```
428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git status
On branch master
nothing to commit, working tree clean

428991@LINL190904638 MINGW64 /d/git-demo (master)
$

428991@LINL190904638 MINGW64 /d/git-demo (master)
$
```



Git

SourceTree

- To visually see every commit and branches, download SourceTree from https://www.sourcetreeapp.com/
- During installation, SourceTree will automatically detect your Git installation
- Once installed, import the git project created or add a remote origin
- As you can see below, during our demo we have made 3 commits to the master branch



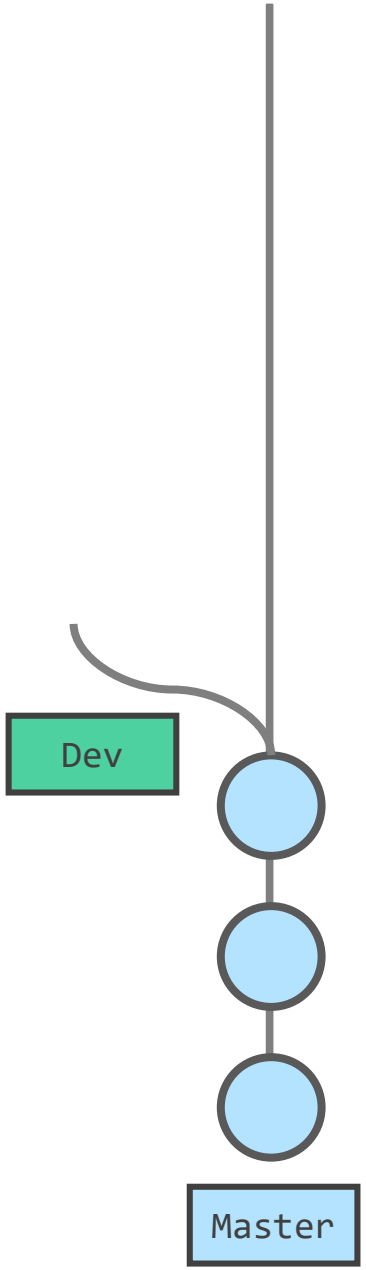
Git

Branches

- Show current branch. The current local branch will be marked with an asterisk (*)
  - `git status`
- List all of the branches in your repository
  - `git branch`
  - `git branch --list`
- Create a new branch dev
  - `git branch dev`
- Switch to another branch
  - `git checkout dev`
- Create and checkout at once
  - `git checkout -b dev`
- Check HEAD (HEAD points out the last commit in the current checkout branch)
  - `git show head`

For making commits to a new branch, you should be in that branch already!

```
428991@LINL190904638 MINGW64 /d/git-demo (dev)
$ git log --oneline
755a2fc (HEAD -> dev, master) Added .gitignore
2fedc59 Second Commit, Added file4.txt and file5.txt
53b6941 Initial Commit

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git status
On branch master
nothing to commit, working tree clean

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git branch
* master

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git branch dev

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git branch
  dev
* master

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git checkout dev
Switched to branch 'dev'

428991@LINL190904638 MINGW64 /d/git-demo (dev)
$ git status
On branch dev
nothing to commit, working tree clean

428991@LINL190904638 MINGW64 /d/git-demo (dev)
$ git branch
* dev
  master

428991@LINL190904638 MINGW64 /d/git-demo (dev)
$ git show head
commit 755a2fc69ec0ef2b740cf5e2b8c8768dc69e20ec (HEAD -> dev, master)

428991@LINL190904638 MINGW64 /d/git-demo (dev)
$ git log --oneline
755a2fc (HEAD -> dev, master) Added .gitignore
2fedc59 Second Commit, Added file4.txt and file5.txt
53b6941 Initial Commit
```

Git

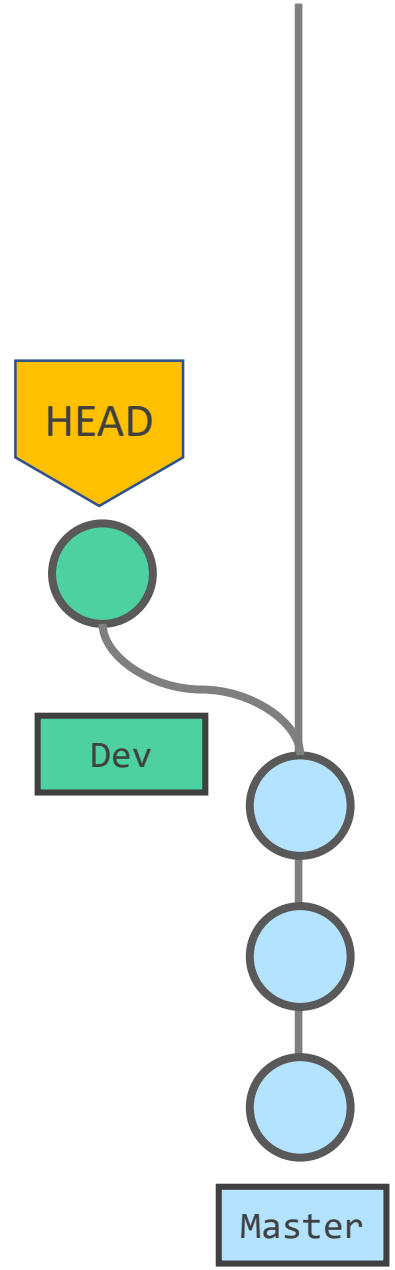
Branches

Push the files to origin

- Add a new file file6.txt ad check the status
  - `git status`
- Add the file to staging
  - `git add .`
- Commit to local
  - `git commit -m "Added Sixth File to Dev Branch"`
- Push to remote dev branch
  - `git push origin dev`
- Check log
  - `git log --oneline`

```
428991@LINL190904638 MINGW64 /d/git-demo (dev)
$ git log --oneline
7f953dd (HEAD -> dev, origin/dev) Added Sixth File to Dev Branch
755a2fc (origin/master, master) Added .gitignore
2fedc59 Second Commit, Added file4.txt and file5.txt
53b6941 Initial Commit
```

```
428991@LINL190904638 MINGW64 /d/git-demo (dev)
$ ls -a
./  ../  .git/  .gitignore  file1.txt  file2.txt  file3.txt  file4.txt
file5.txt  TODO.txt

428991@LINL190904638 MINGW64 /d/git-demo (dev)
$ touch file6.txt

428991@LINL190904638 MINGW64 /d/git-demo (dev)
$ echo "My Sixth File" > file6.txt

428991@LINL190904638 MINGW64 /d/git-demo (dev)
$ git status
On branch dev
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file6.txt

nothing added to commit but untracked files present (use "git add" to track)

428991@LINL190904638 MINGW64 /d/git-demo (dev)
$ git add .
warning: LF will be replaced by CRLF in file6.txt.
The file will have its original line endings in your working directory

428991@LINL190904638 MINGW64 /d/git-demo (dev)
$ git commit -m "Added Sixth File to Dev Branch"
[dev 7f953dd] Added Sixth File to Dev Branch
 1 file changed, 1 insertion(+)
 create mode 100644 file6.txt

428991@LINL190904638 MINGW64 /d/git-demo (dev)
$ git push origin dev
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 295 bytes | 295.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'dev' on GitHub by visiting:
remote:      https://github.com/kunchalavikram1427/test-repo/pull/new/dev
remote
To https://github.com/kunchalavikram1427/test-repo.git
 * [new branch]      dev -> dev
```



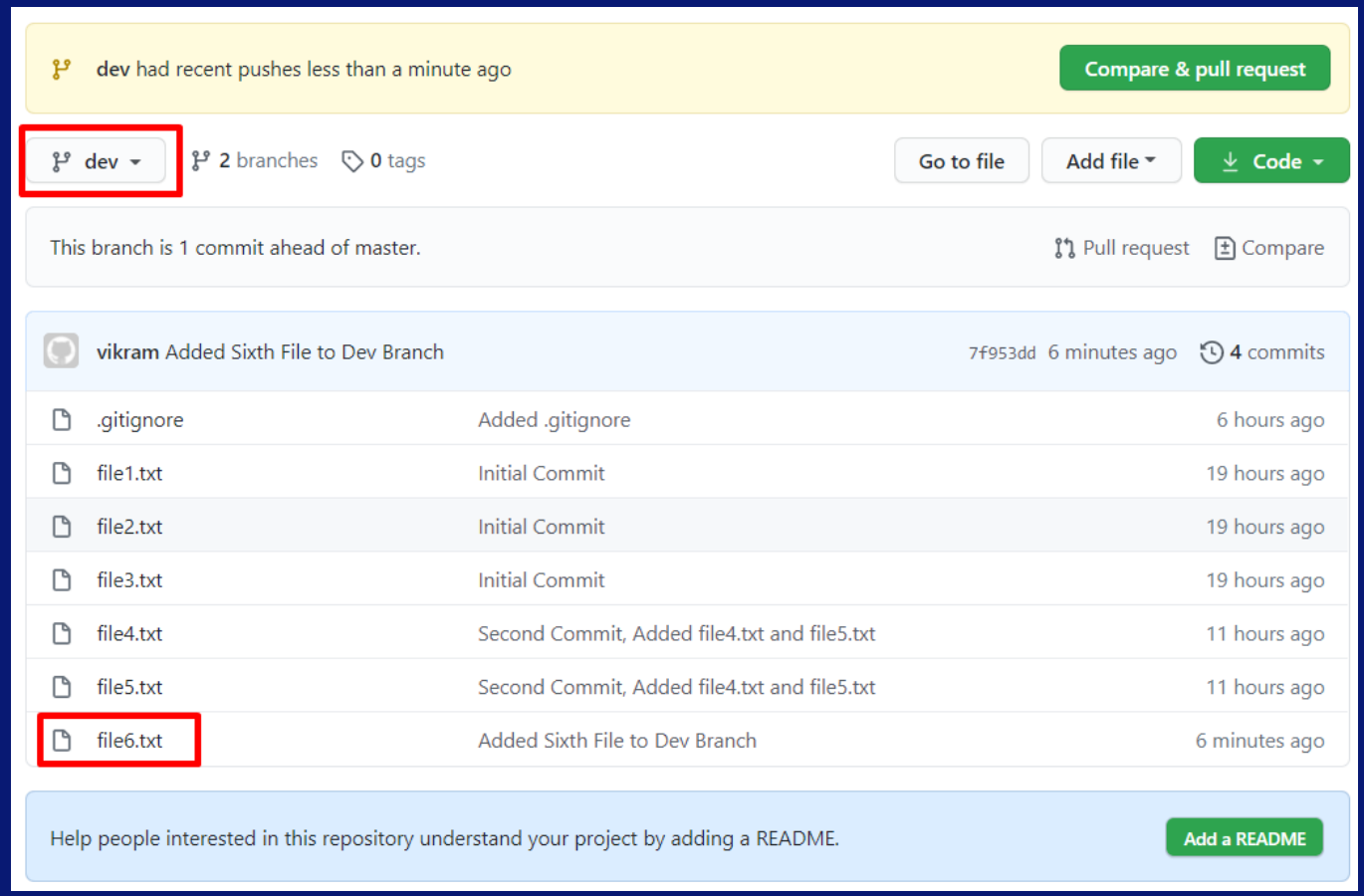
git:kunchalavikram1427

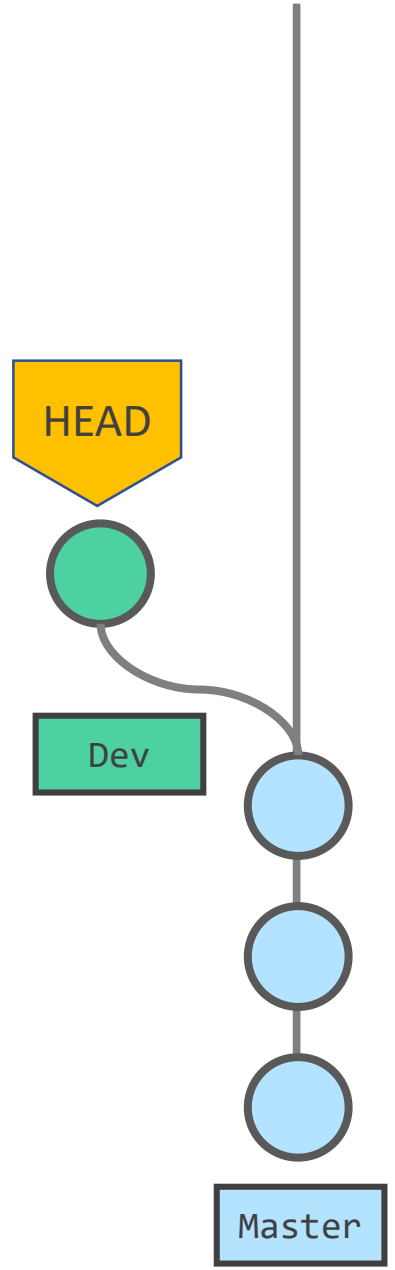
Git

Branches

Push the files to origin

- Check your files in the repository in dev branch



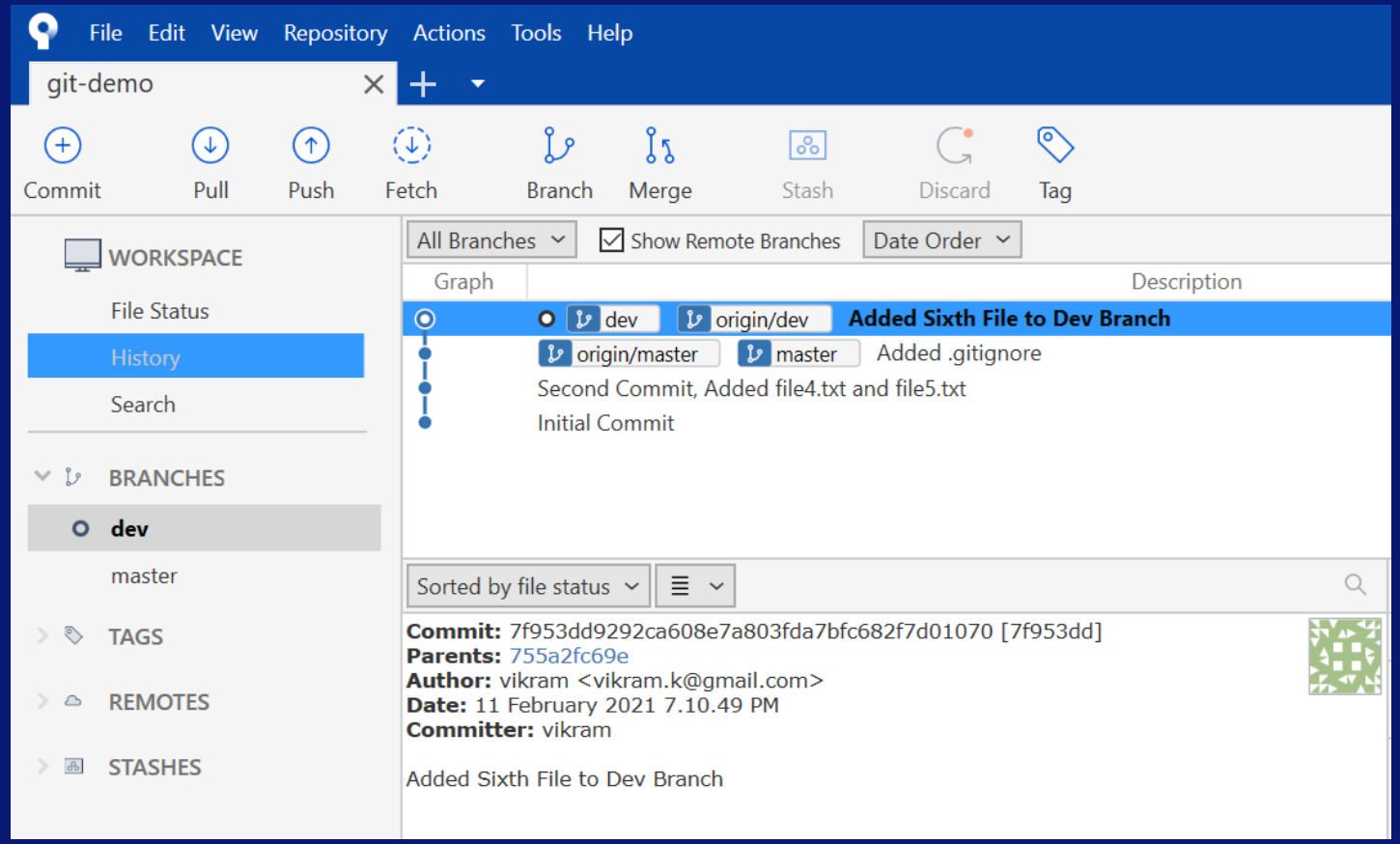


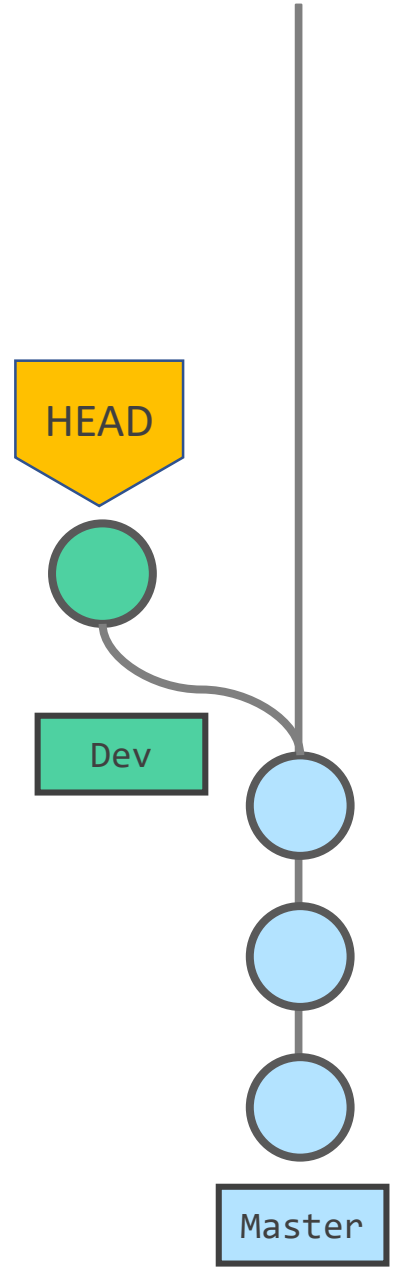
Git

Branches

Push the files to origin

- Verify the same in SourceTree





git:kunchalavikram1427

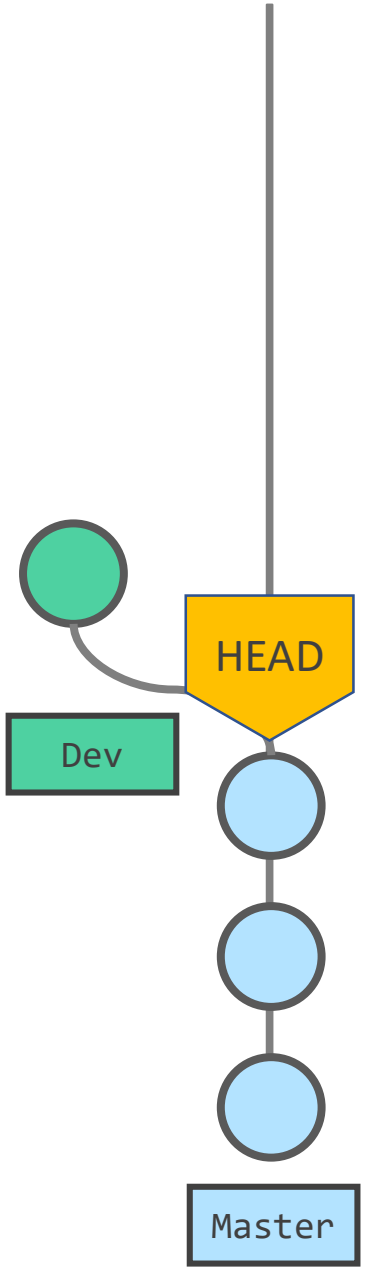
Git

Branches

Checkout

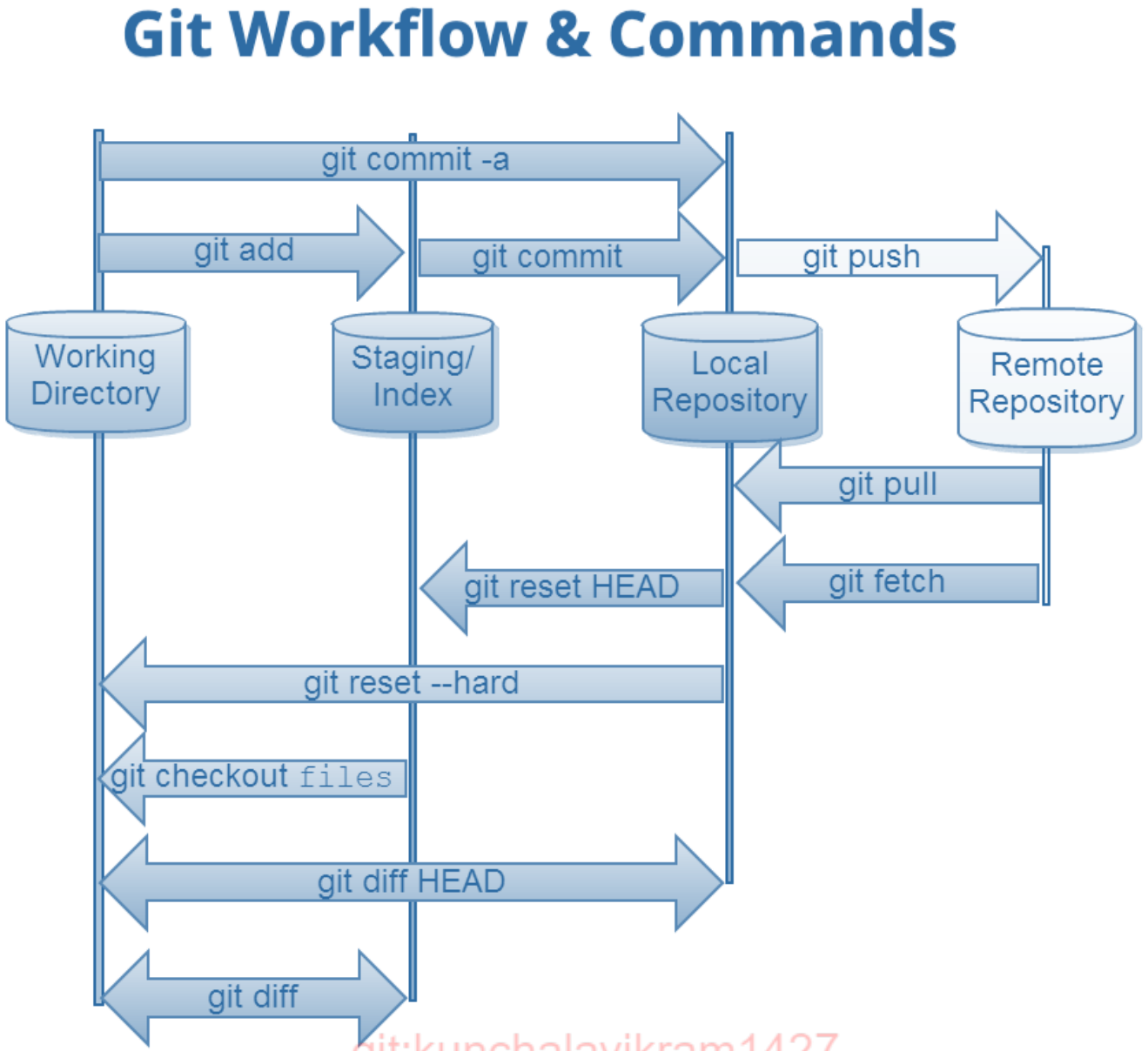
- The git checkout command operates upon three distinct entities: files, commits, and branches
- The git checkout command lets you navigate between the branches created by git branch
- Checking out a branch updates the files in the working directory to match the version stored in that branch, and it tells Git to record all new commits on that branch
  - `git checkout master` – switches the branch to master and update the files as well
- Since file6.txt is only in dev branch, the file is missing in the master branch

```
428991@LINL190904638 MINGW64 /d/git-demo (dev)
$ ls -a
./  ../  .git/  .gitignore  file1.txt  file2.txt  file3.txt  file4.txt  file5.txt  file6.txt  TODO.txt

428991@LINL190904638 MINGW64 /d/git-demo (dev)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ ls -a
./  ../  .git/  .gitignore  file1.txt  file2.txt  file3.txt  file4.txt  file5.txt  TODO.txt

428991@LINL190904638 MINGW64 /d/git-demo (master)
$ git show head
commit 755a2fc69ec0ef2b740cf5e2b8c8768dc69e20ec (HEAD -> master, origin/master)
```

Git

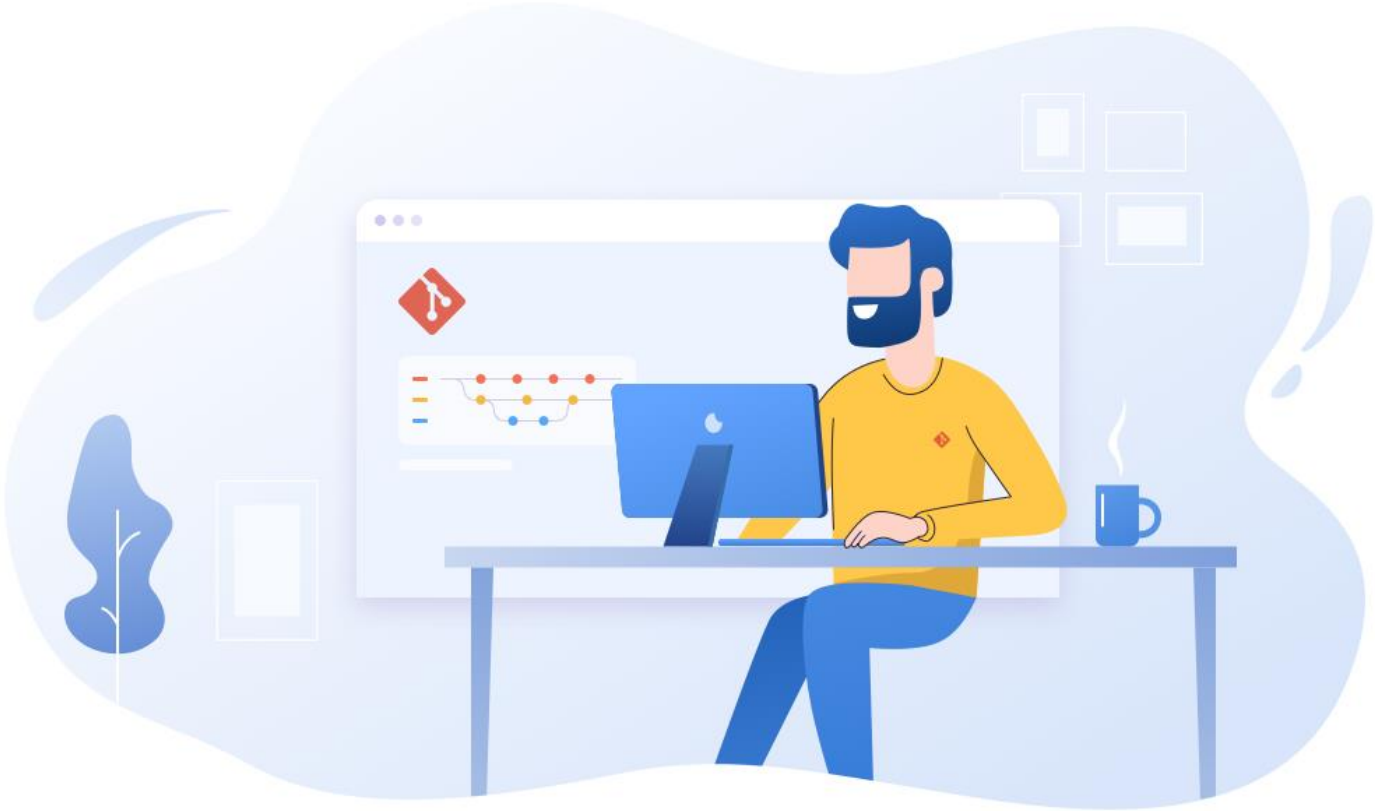
Workflow




git:kunchalavikram1427
git
PART - 02
Git Pull, Fetch, Clone, Reset, Merge
Conflicts, Cherry pick etc.,
git:kunchalavikram1427
/kunchalavikram1427

References

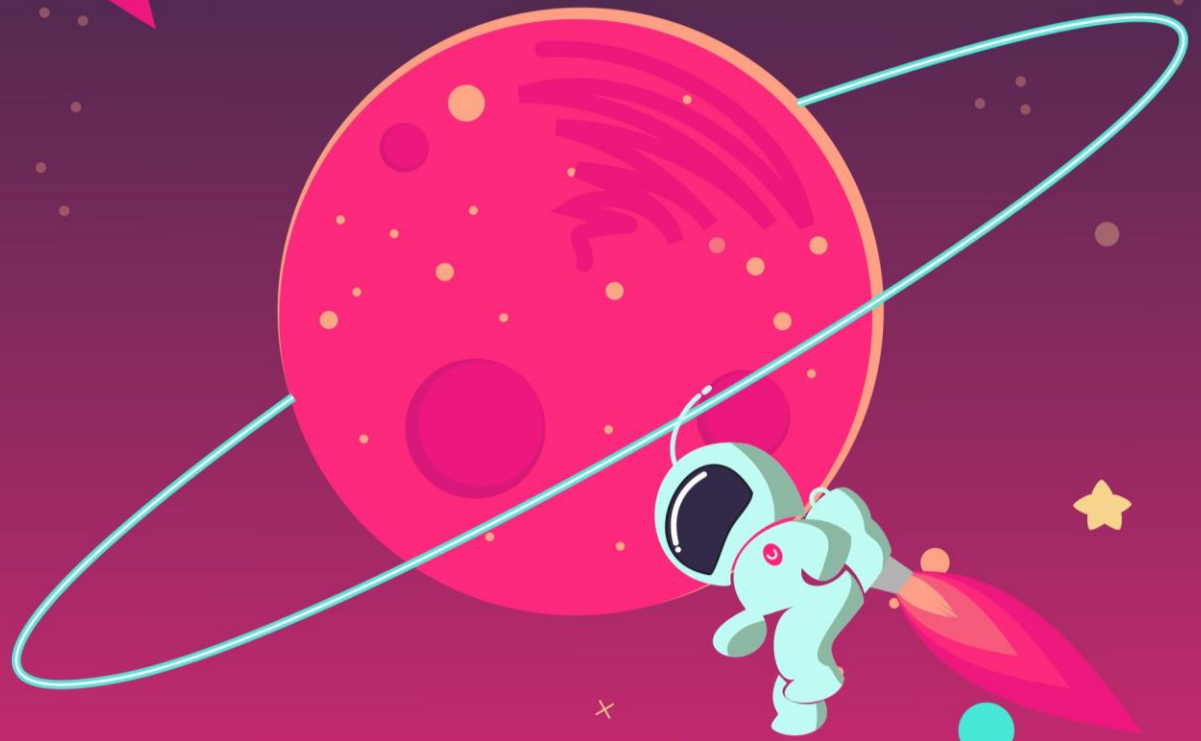
- https://git-scm.com/book/en/v2/Git-Tools-Reset-Demystified



Subscribe to my Facebook page:
https://www.facebook.com/vikram.devops
and join my group:
https://www.facebook.com/groups/171043094400359
for all latest updates on DevOps, Python and IoT

https://www.youtube.com/channel/UCE1cGZfooxT7-VbqVbuKjMg

Thank You



git:kunchalavikram1427