IT Club

GIT WORKSHOP

Day 1: Introduction to Git and VCS

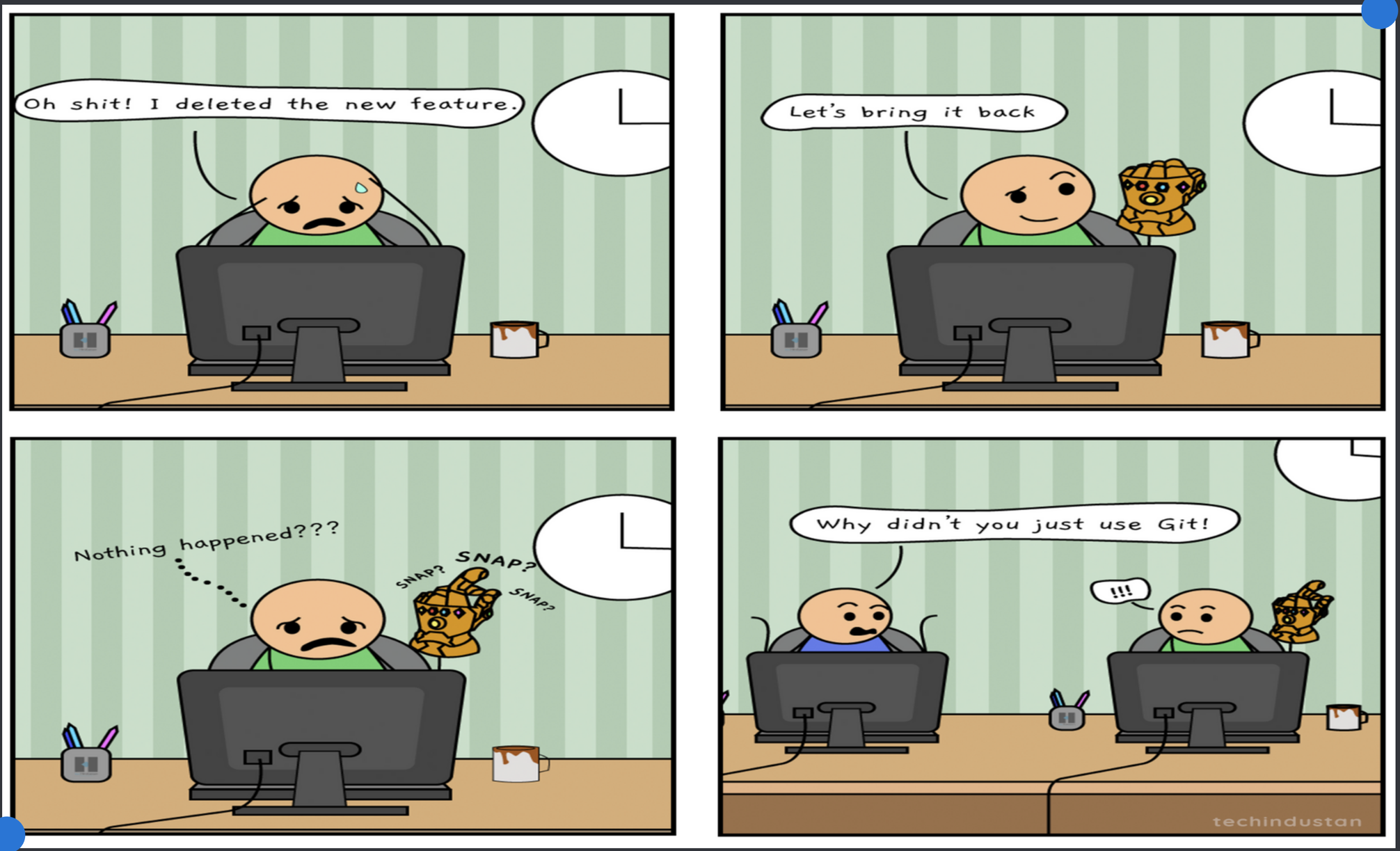
Prerequisites

IT Club

- Install Git on your computer.
- Create a Github account.
- Install IDE (VS Code recommended).

Part 1

The Need of Version Control


Oh shit! I deleted the new feature.
Let's bring it back
Nothing happened???
SNAP?
SNAP?
SNAP?
Why didn't you just use Git!
!!!
techindustan

Version Control Systems

IT Club

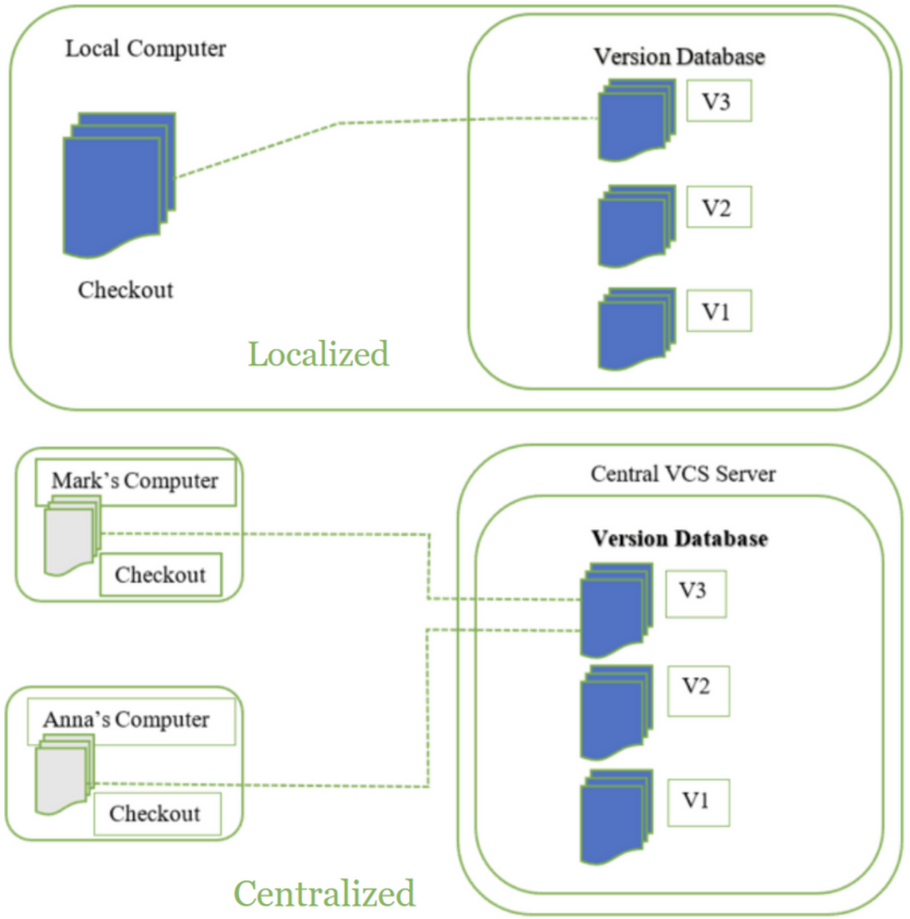
Version control systems (VCSs) are tools used to track changes to source code (or other collections of files and folders).

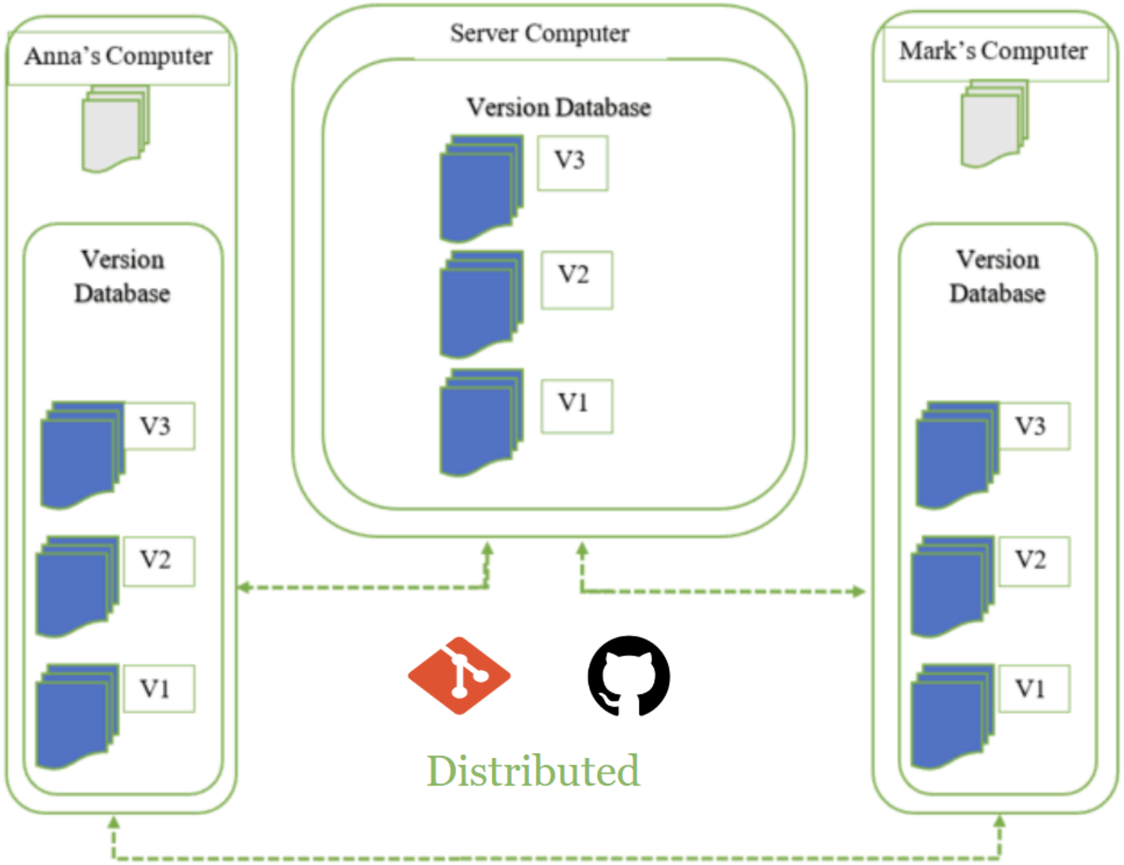
Version Control Systems

IT Club

- **managing changes:** versions of files over time
- **collaborate:** multiple users to on a project
- **history:** track of the changes made
- **revert:** go back to previous versions
- **conflict management:** multiple users working on same files
- **branching:** multiple features at the same time

Types


Local Computer
Version Database
V3
V2
V1
Checkout
Localized
Mark's Computer
Checkout
Central VCS Server
Version Database
V3
V2
V1
Anna's Computer
Checkout
Centralized


Anna's Computer
Version Database
V3
V2
V1
Server Computer
Version Database
V3
V2
V1
Mark's Computer
Version Database
V3
V2
V1
Distributed

Heard of these?

IT Club

Distributed

Centralized

Heard of these?

IT Club

Centralized

And of course

GIT

Distributed

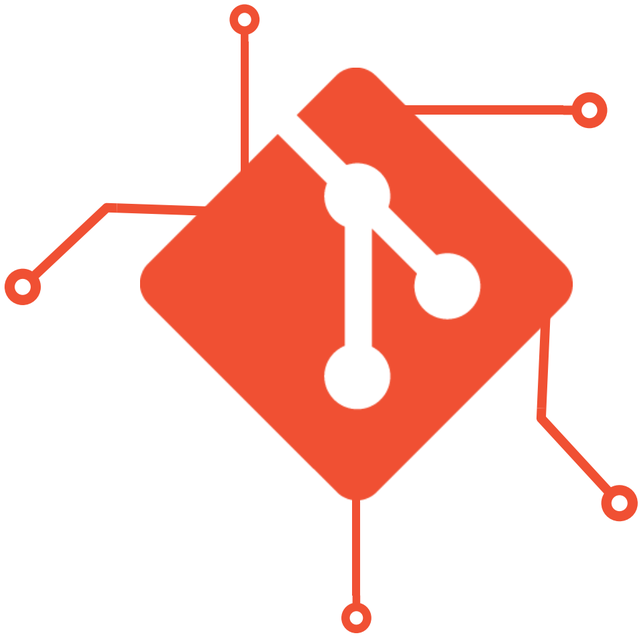
mercurial

Git it?

Fully Distributed

Speed

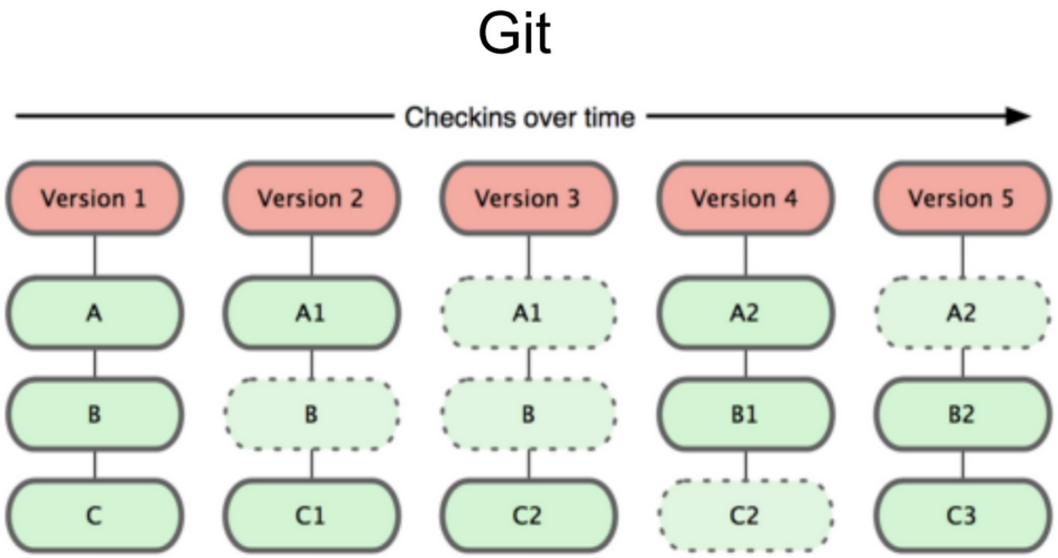
Open Source



Large Projects

Non Linear Development

Git Snapshots



Reduntant but fast

Snapshots of entire state

Part 2

Gitting Started

Installation

Setting Up Git

Checking installation of git

```
$ git version
```

Setting Up Git

IT Club

Configuring Git with username and email

`git config --global user.name “[firstname lastname]”`
set a name that is identifiable for credit when review version history
`git config --global user.email “[valid-email]”`
set an email address that will be associated with each history marker

Example:

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Checking your Settings

IT Club

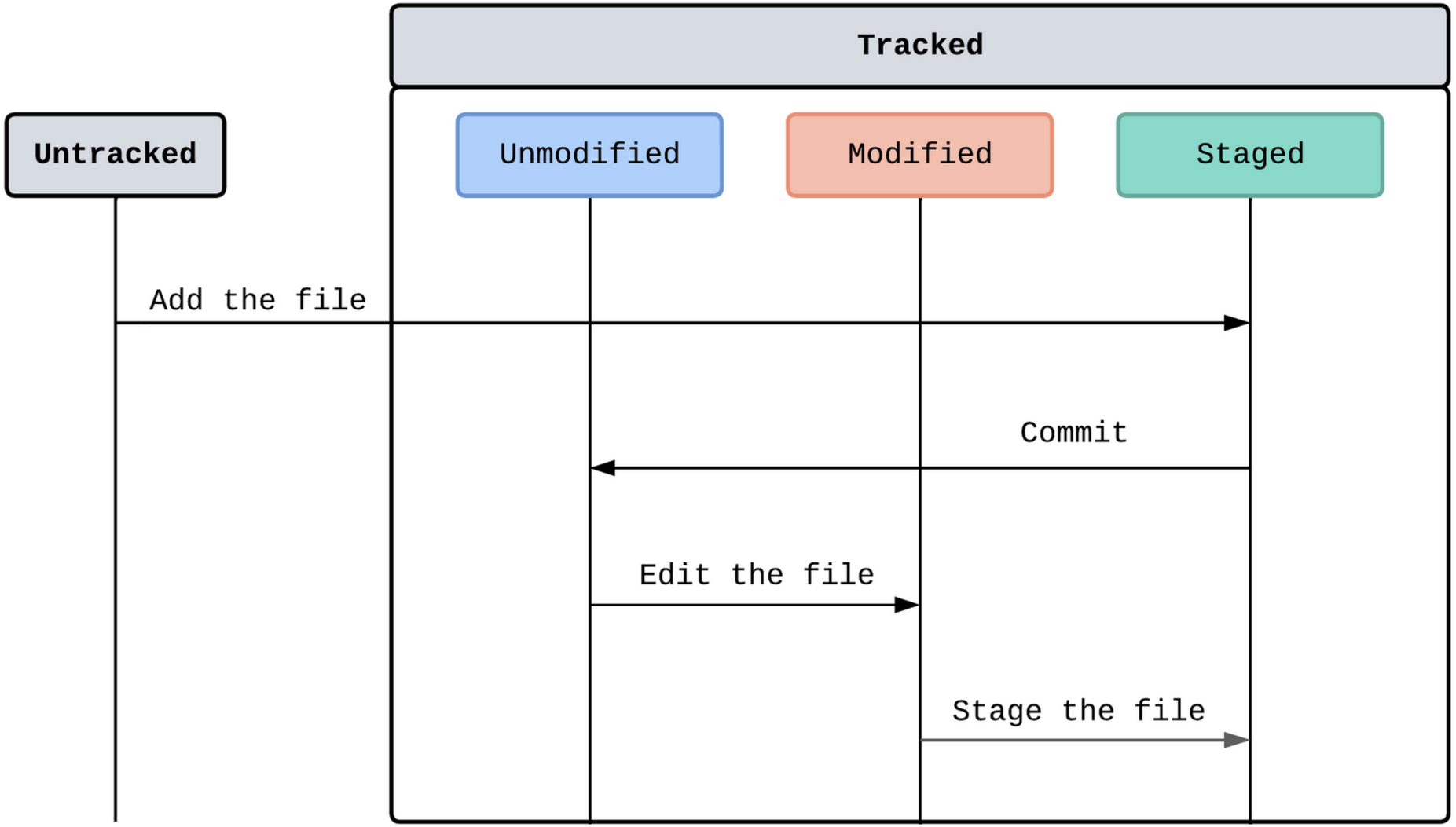
```
$ git config --list
```

list all the settings Git can find at that point

```
$ git config user.name
```

check specific key value for a setting

Lifecycle of status of file



Setting Up Git

IT Club

Creating a new git repository

```
git init
```

initialize an existing directory as a Git repository

or you can clone a repository but we'll get into it later.

- A hidden folder named **.git** is created which contains all the information of the repository such as change history, settings, compressed version of each file, etc.

Tracking your files

IT Club

```
$ git add <filepath>
```

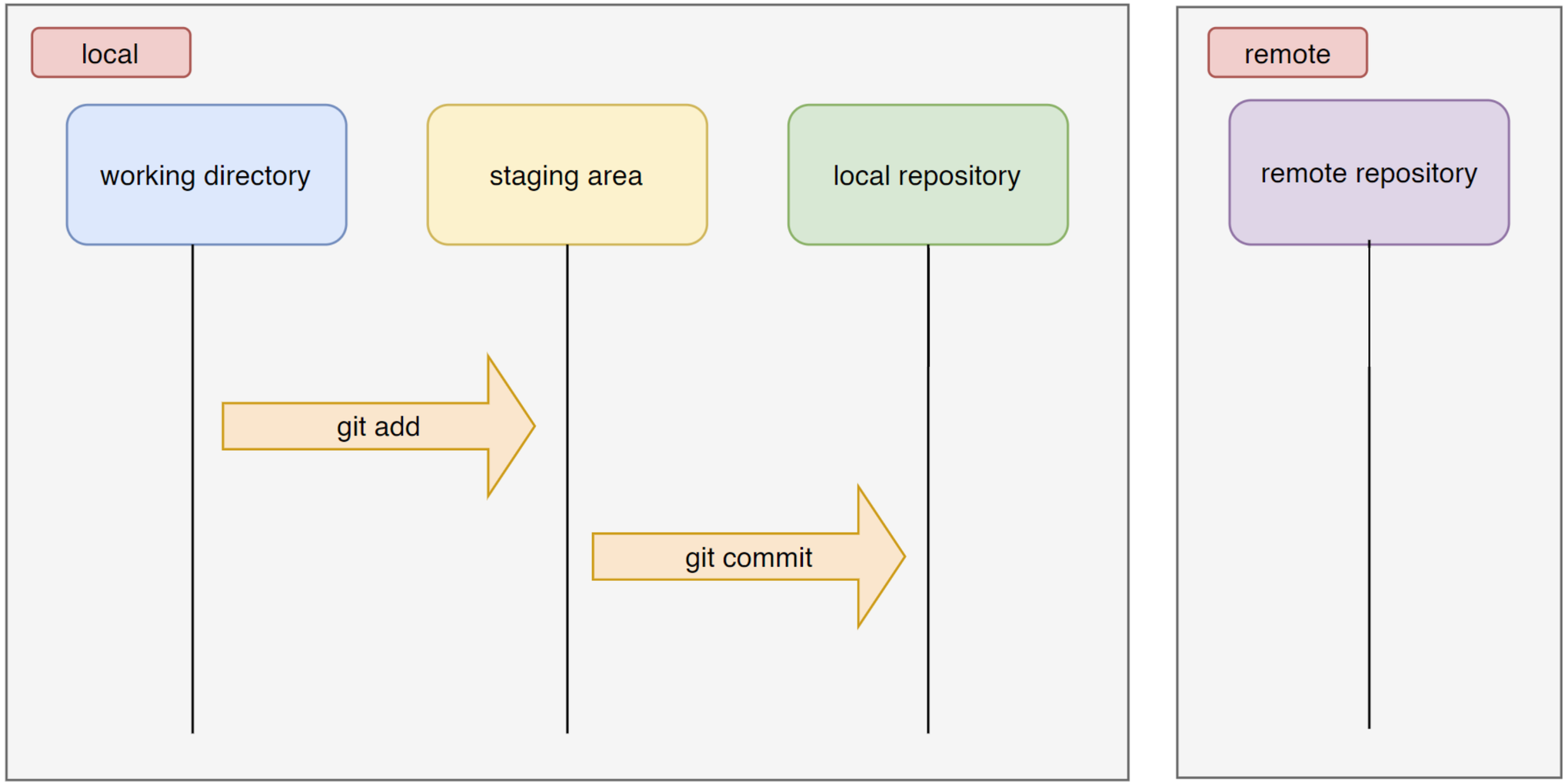
- Stage ***changes*** in the specified file(s), preparing them for the next commit
- Place the ***changes*** to the so called "staging area"

Example:

```
$ git add hangman.c
```

Git Workflows

IT Club



Checking the Status of Your Files

IT Club

```
$ git status
```

- Shows the current status of your git repo
- Displays the information about tracked and untracked files in the current working directory
- Tracked files are files that Git is aware of and are already being version controlled.
- Untracked files are files that are not yet added to the Git repository.

Batch staging

```
$ git add .
```

- Stage ***changes*** in all the files in the current directory and its subdirectories.
- where **.** specifies the current working directory

```
$ git add *.c
```

- Stage ***changes*** in all files with the ".c" extension in the current directory and its subdirectories.

Some more batch staging

```
$ git add file1.txt file2.c file3.py
```

Stage ***changes*** in "file1.txt", "file2.c", and "file3.py".

```
$ git add my_folder/
```

Stage ***changes*** in an "entire" folder.

Taking snapshots

IT Club

```
git commit -m “[descriptive message]”
```

commit your staged content as a new commit snapshot

- **Atomic commits** - It's a best practice to make commits atomic, meaning each commit should represent a single logical change

- The commit message should be ***descriptive***, explaining the changes made in the commit.

Viewing Changes

```
$ git diff
```

- Shows the changes between the working directory and the staging area (or the last commit)
- Displays the differences in a line-by-line format, highlighting additions and deletions with "+" and "-" signs, respectively.

Viewing the Commit History

IT Club

```
$ git log
```

- Displays the list of commits in reverse chronological order, showing the latest commits first
- Each commit in the log includes information such as the commit hash (SHA-1 checksum), author name, author email, commit date, the commit message, and the commit description (if any).
- Pressing the **Enter** key scrolls down through the log, displaying more commits if available. Press **q** to exit the log view.

Some useful commands

Undoing things with git restore

IT Club

```
$ git restore <filepath>
```

- discard changes made to a specific file in the working directory and revert it to the state of the last commit
- **Important:**
  Don’t ever use this command unless you absolutely know that you don’t want those unsaved local changes.
  Please use with caution.

```
$ git restore hangman.c
```

- Revert changes in *hangman.c* to the last commit

Some useful commands

IT Club

```
$ git diff --staged
```

- shows the changes between your staged changes and your last commit
- used when we want to see what we've staged that will go into our next commit

Some useful commands

```
$ git diff --name-only
```

- shows only the names of the files which have changed between the working directory and the staging area (or the last commit)

- useful when you only need to know which files have been modified, added, or deleted, without showing the actual content changes.

Some useful commands

IT Club

```
$ git reset
```

- Move staged changes back to the working directory.
- can be used when we mistakenly stage (add) files to the staging area
- The **git reset** command comes with different options that can be used to achieve different outcomes, which we'll discuss in later days of the workshop.

Limiting Log Output

```
$ git log --oneline
```

- Shows each commit as a single line, displaying only the abbreviated commit hash and the first line of the commit message.

`$ git log -<n>` Show only the last n commits.

Example:

`$ git log -5` Show only the last 5 commits.

Getting Help

`$ git help`

- To get general help and see a list of common Git commands

`$ git help <command>`

- To get help for specific Git command

Example:

`$ git help log` To get help for the command **git log**

Questions?

Quiz Time

Who is the original author of Git?

Linus Torvalds

Principal author of the largest open-source OS Linux

Which hashing functions does Git use?

Answer:

SHA-1 hash

SHA-256 hash

b1f87a50fcacaa0ba9c6f3ca79fcf11e96c6d2

How can you limit number of commits displayed with git log command?

A. Use the "--max-commits" flag followed by the desired number.

B. Include the "-<n>" option where `n` is the desired number.

C. Use the "git log --limit" command.

D. There is no way to limit the number of commits displayed.

Answer:

B

Thank you