Introduction, Installation and Basic Commands

GIT
Git is distributed version control system that allows multiple users to work on a project simultaneously without interfering with each other's changes.

Key Features of Git
- ***Version Control***: tracks and manages changes to files over time.
- ***Distributed Architecture***: Every user has a complete copy of the repository, including its history, which allows for offline work.
- ***Branching and Merging***: Enables users to create branches for new features or fixes and later merge then back into the main project.
- ***Staging Area***: A place where changes can be reviewed before committing them to the project.
- ***Commit History***: Keeps a detailed history of changes, making it easy to track progress and revert to earlier versions if needed.

Common Terminology:
- ***Repository(Repo)***: A storage space for your project, which can be local or remote.
- ***Commit***: A snapshot of changes made to files in the repository.
- ***Branch***: A separate line of development, allowing for concurrent work on different features or fixes.
- ***Merge***: Combining changes from one branch to another.
- ***Clone***: A copy of a repository, created from a remote repository to your local machine.

Installation & GUIs

- **GitHub for Windows***
  https://windows.github.com

Setup
Configuring user information used across all repositories.

1. set name that is identifiable for credit when review version history.

```
git config --global user.name "[firstname lastname]"
```

2. set email address that will be associated with each history marker.

```
git config --global user.email "[valid-email]"
```

3. set automatic command line coloring for Git for easy reviewing

```
git config --global color.ui auto
```

Setup & init
Configuring user information, initializing and cloning repositories.

1. initializing an existing directory as a Git repository.

```
git init
```

2. retrieve an entire repository from a hosted location via URL

```
git clone [url]
```

Stage & Snapshot
Working with snapshots and the Git staging area.

1. show modified files in working directory, staged for your next commit

```
git status
```

2. add a file as it looks now to your next commit

```
git add [file]
```

3. unstage a file while retaining the changes in the working directory

```
git reset [file]
```

4. diff of what is changed but not staged

```
git diff
```

5. diff of what is staged but nor committed

```
git diff --staged
```

6. commit your staged content as a new commit snapshot

```
git commit -m "[descriptive message]"
```