**Git** is not Github. Git is the version control software, and Github is a git repository hosting service which offers all the source code management provided in git. Github is where you upload your git repository.

**Centralized Version Controlling**  
The systems such as CVS, Subversion, and Perforce have a single server that contains all the versioned files, and a number of clients that check out files from that central place.

**Distributed Version Controlling**  
In a DVCS (such as Git), clients don’t just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history. Thus, if any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to  
restore it. Every clone is really a full backup of all the data.

**Comparison**

Flexibility: DVCS offers more flexibility as developers can work offline and commit locally. CVCS requires constant network access.

Resilience: DVCS is more resilient to failures due to the distributed nature of the repositories.

Complexity: DVCS can be more complex to manage, with multiple repositories to synchronize, while CVCS is simpler with just one central repository.

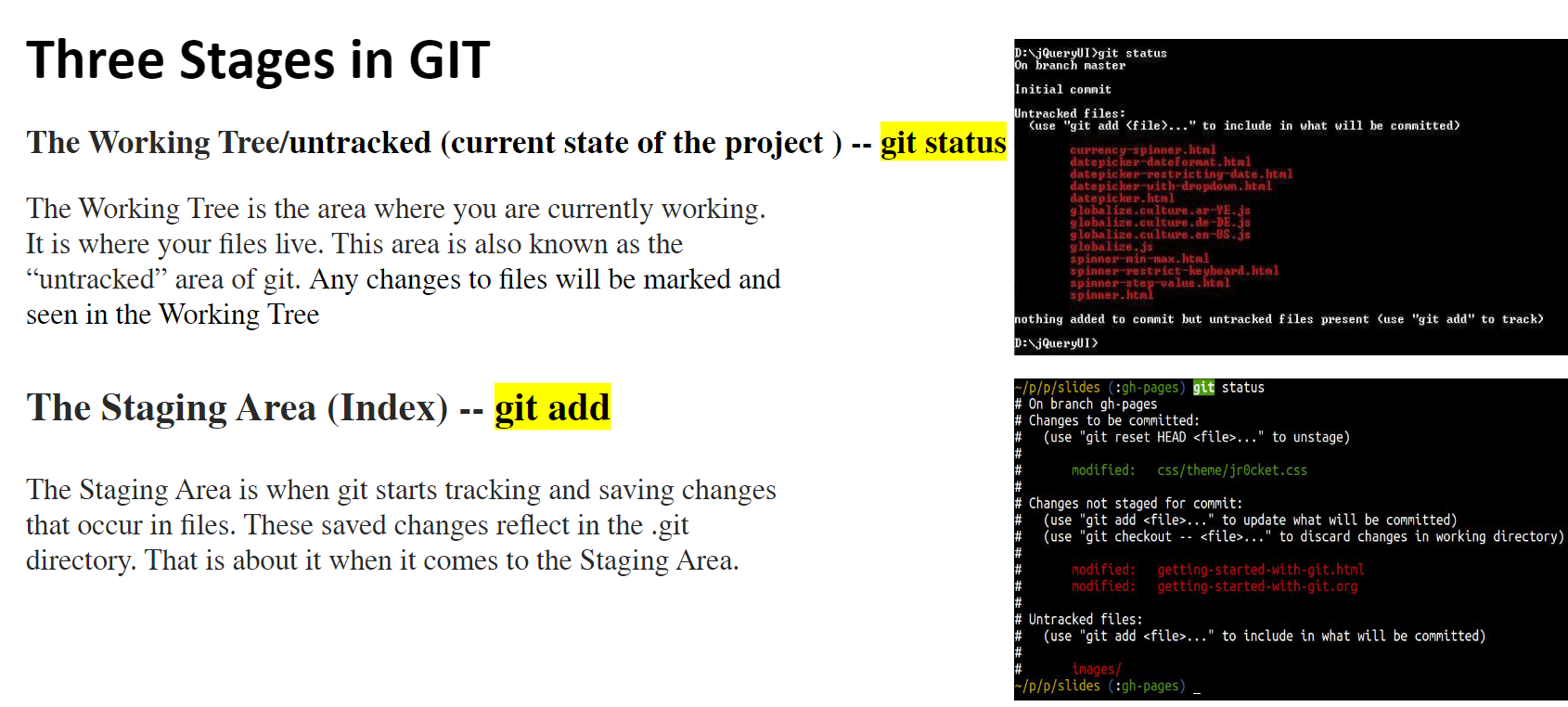
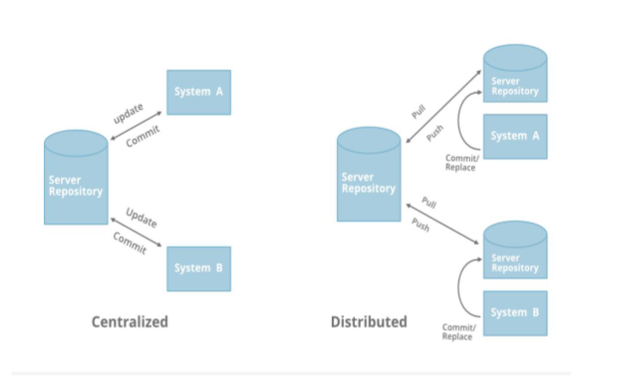
Git, as a DVCS, has become the standard for modern software development due to its flexibility and resilience.

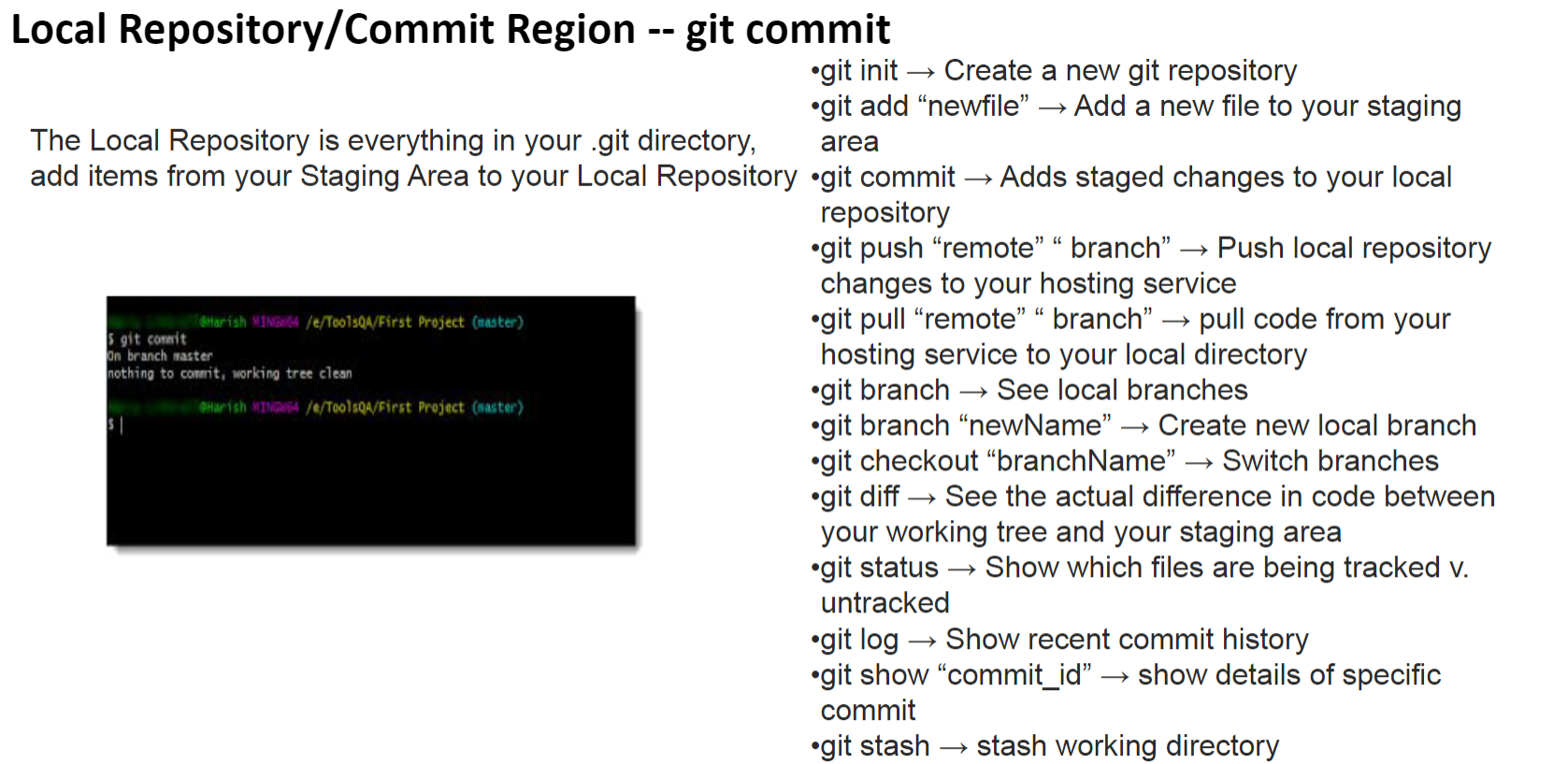
**Where we can use centralized**

in legacy system, Teams with Simple Needs like

Small Teams: Small development teams that don't require the advanced branching and merging capabilities of a DVCS might find CVCS easier to manage.

Less Complexity: Teams working on simple projects with linear workflows might not need the additional features of DVCS.





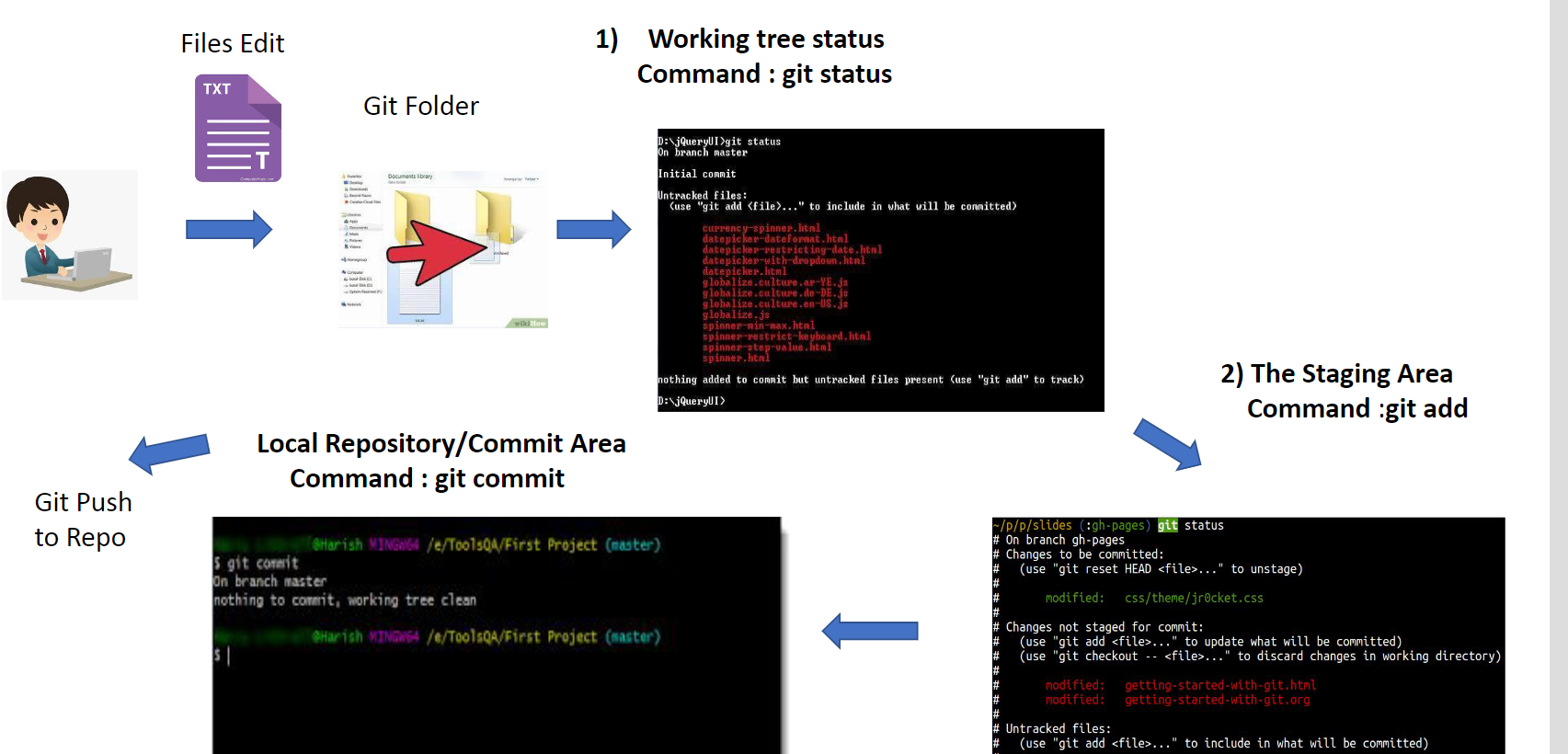
**Workflow Summary**

Working Directory: You modify files as needed.

Staging Area: You stage the changes you want to include in the next commit.

Repository: You commit the staged changes, creating a snapshot in the project history.

This three-stage process allows for precise control over what changes are recorded and provides the ability to manage and review changes before they become part of the project's history.



**Installation of GIT:**

Installing Git depends on your operating system. Here’s a step-by-step guide for installing Git on Windows.

**1. Installing Git on Windows**

**Step 1: Download Git**

* Visit the official Git website: [Git for Windows](https://git-scm.com/download/win).
* The download should start automatically. If not, click the appropriate link to download the installer.

**Step 2: Install Git**

* Run the downloaded .exe file.
* Follow the installation prompts:
  + **Adjusting your PATH environment**: Choose "Git from the command line and also from 3rd-party software."
  + **Choosing the SSH executable**: Use the built-in OpenSSH.
  + **Choosing HTTPS transport backend**: Use the OpenSSL library.
  + **Configuring the line ending conversions**: Choose "Checkout Windows-style, commit Unix-style line endings."
  + **Extra options**: Select the default options unless you have specific requirements.

**Step 3: Verify Installation**

* Open the Command Prompt or Git Bash (installed along with Git).
* Type git --version and press Enter.
* You should see the installed version of Git.

**Post-Installation Configuration**

After installing Git, you should configure your user information. Run these commands in your terminal:

git config --global user.name "Your Name"

git config --global user.email "your.email@example.com"

This information will be associated with your commits.

**Branching in Git is a powerful feature that allows you to create separate lines of development within your project. Each branch is an independent version of your codebase that you can work on without affecting the main project. This is particularly useful for developing new features, fixing bugs, or experimenting with changes in isolation from the main code.**

**Key Concepts of Git Branching**

1. **Branch: A branch in Git is essentially a pointer to a commit. It allows you to work on different versions of a project simultaneously.**
2. **Main Branches:**
   * **main or master: The default branch that is created when you initialize a new Git repository. It typically represents the stable version of your code.**
   * **Feature Branches: These are branches created to work on new features, bug fixes, or any other tasks that require separate development.**

**Basic Branch Operations in Git**

**1. Creating a New Branch**

* **To create a new branch, use the following command:**
* **git branch <branch\_name>**
* **This creates a new branch but does not switch to it. The branch will point to the current commit.**

**2. Switching to a Branch**

* **To switch to an existing branch, use:**

**git checkout <branch\_name>**

* **You can create and switch to a new branch in one command:**

**git checkout -b <branch\_name>**

**3. Merging Branches**

* **Once you've completed work on a branch, you can merge it back into the main (or any other) branch:**

**git checkout main**

**git merge <branch\_name>**

* **This command will integrate the changes from <branch\_name> into the main branch.**

**4. Deleting a Branch**

* **After a branch is merged or if you no longer need it, you can delete it:**

**git branch -d <branch\_name>**

* **To force-delete a branch (e.g., if it hasn’t been merged):**

**git branch -D <branch\_name>**

**5. Listing Branches**

* **To list all branches in your repository:**

**git branch**

* **The current branch will be highlighted with an asterisk (\*).**

**Branching Strategies in Depth**

Git Flow:

Main Branches: master (or main) for production-ready code, and develop for the latest development changes.

Feature Branches: Created from develop for new features. Merged back into develop when complete.

Release Branches: Created from develop when preparing a new production release. Final testing and bug fixes occur here.

Hotfix Branches: Created from master to address critical bugs in production. Merged back into both master and develop.

GitHub Flow:

Main and Feature Branches: Simple strategy with just a main branch for production and short-lived feature branches for each task. Feature branches are merged into main through pull requests after review.

Trunk-Based Development:

Single Branch: Developers commit directly to the main branch frequently, often several times a day. Feature flags or toggles are used to manage incomplete features without breaking the production code.