**30-MINUTE PRESENTATION AMONG THREE SPEAKERS**

* Each speaker covers about **10 minutes**.
* Each speaker has a clear and natural handoff to the next.
* Each section feels coherent (no abrupt jumps).

**Speaker 1: Introduction and Fundamentals (10 minutes)**

**Slides:**

1. Title Slide
2. Introduction - Why Software Maintenance Matters
3. What is Software Maintenance?
4. Types of Software Maintenance
5. Software Maintenance Activities
6. Why Version Control is Essential

**Main Role:**

* Set the stage.
* Create urgency and interest.
* Lay down the basics.

**Speaker Handoff Line:**  
*"Now that we understand why maintenance is crucial, [Speaker 2] will show you the real tools that make maintenance work — BitBucket and GitLab."*

**Speaker 2: Tools and Practical Setup (10 minutes)**

**Slides:**  
7. BitBucket Overview  
8. GitLab Overview  
9. BitBucket vs GitLab  
10. Hands-On: Repository Setup  
11. Implementing Branching Strategies  
12. Code Review and Merge Requests  
13. Bug Tracking and Issue Management

**Main Role:**

* Dive into practical tools.
* Highlight hands-on setup and workflows.

**Speaker Handoff Line:**  
*"With tools and workflows in place, [Speaker 3] will now guide you through making everything smarter, faster, and safer — with automation, security, and real-world examples."*

**Speaker 3: Automation, Security, Best Practices and Conclusion (10 minutes)**

**Slides:**  
14. Setting up CI/CD Pipelines  
15. Automating Security & Compliance  
16. Best Practices for Software Maintenance  
17. Case Study: Real World Example  
18. Conclusion: Key Takeaways

**Main Role:**

* Show automation and security features.
* Emphasize best practices.
* Inspire action and wrap up.

**Closing Tip:**  
End with a strong call for continuous improvement:  
*"Maintenance is not an afterthought — it’s the key to resilient, world-class software."*

Summary

|  |  |  |  |
| --- | --- | --- | --- |
| **Speaker** | **Time** | **Focus** | **Slides** |
| Speaker 1 | 0–10 mins | Introduction + Basics | 1–6 |
| Speaker 2 | 10–20 mins | Practical Tools and Setups | 7–13 |
| Speaker 3 | 20–30 mins | Automation + Best Practices + Wrap | 14–18 |

**Create a new repository on the command line**

echo "# software-Dev-project" >> README.md

git init

git add README.md

git commit -m "first commit"

git branch -M main

git remote add origin https://github.com/e-las/software-Dev-project.git

git push -u origin main

**Push an existing repository from the command line**

git remote add origin https://github.com/e-las/software-Dev-project.git

git branch -M main

git push -u origin main

Generated token for GitHub account

ghp\_kRlNPjEscXw1wJZnvj4Bv5WVvON98u4fS2UL

To connect your **local machine to GitHub**,

Using **HTTPS** or **SSH**.

Guideline for both methods:

**How to Connect Local Machine to GitHub**

**1. Install Git (if not already)**

Download and install from:  
 <https://git-scm.com/downloads>  
After install, open a terminal (PowerShell or Git Bash) and check:

git --version

**2. Authenticate Git with GitHub**

**Option A: Using HTTPS**

1. Go to your GitHub repo, click **Code** → select **HTTPS** URL.
2. Clone it:
3. git clone https://github.com/username/repo.git
4. The first push will ask for your **GitHub username** and a **Personal Access Token** (not your password).  
   Create token at: <https://github.com/settings/tokens>

How to generate a **Personal Access Token (PAT)** on GitHub, which replaces your password when using Git over **HTTPS**:

**How to Generate a GitHub Personal Access Token (PAT)**

**Step-by-Step:**

1. **Log in to GitHub**
   * Go to [https://github.com](https://github.com/)
2. **Go to Settings**
   * Click your profile picture (top-right corner) → **Settings**
3. **Access Developer Settings**
   * Scroll down the left menu → Click **"Developer settings"**
4. **Select Personal Access Tokens**
   * Click **"Personal access tokens"** → then **"Tokens (classic)"**
   * Click **"Generate new token"** → **"Generate new token (classic)"**
5. **Fill Token Details**
   * Name your token (e.g., “My Laptop Token”)
   * Set **expiration** (e.g., 30 or 90 days)
   * Select scopes:
     + repo (for full control of private/public repositories)
     + workflow (if you use GitHub Actions)
     + user, gist, etc. (as needed)
6. **Generate Token**
   * Click **"Generate token"**
   * **Copy the token immediately** — you won’t see it again!

**Use Your Token**

When Git asks for your **username**, enter your GitHub username.  
When it asks for a **password**, paste the **token** instead.

Generated token for GitHub account

ghp\_kRlNPjEscXw1wJZnvj4Bv5WVvON98u4fS2UL

**Option B: Using SSH (recommended for regular use)**

1. Generate an SSH key (if you haven’t):
2. ssh-keygen -t ed25519 -C "your\_email@example.com"

(Press enter through defaults)

1. Add the SSH key to your GitHub:
   * Copy your public key:
   * type $env:USERPROFILE\.ssh\id\_ed25519.pub # On PowerShell
   * Go to **GitHub → Settings → SSH and GPG keys → New SSH key**
   * Paste the key and save.
2. Test connection:
3. ssh -T git@github.com
4. Clone using SSH:
5. git clone git@github.com:username/repo.git

**3. Set Your Git Identity (Once)**

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

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