Introduction

This guide provides a quick summary of some common Git commands and procedures, and is not meant to be a comprehensive guide to all things Git. It is simply a list of steps that I have either used so many times I know it by heart, or have to revisit sometimes even while using Git regularly. If you encounter problems or want to explore more advanced topics, there are many excellent tutorials available online, and resources like Stack Overflow can be very helpful for commonly faced issues.

1 Creating a GitHub Account

1. Go to the GitHub website: https://github.com/.
2. Click on the **Sign up** button.
3. Follow the instructions to create your account.
4. After creating your account, verify your email address.

2 Configuring Git with Your Email and Username

Before you start using Git, it's important to configure it with your email address and username, as these will be associated with your commits.

1. Open a terminal window.
2. Set your Git username by typing:

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

   Replace `"Your Name"` with your actual name.
3. Set your email address by typing:

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

   Replace `"your.email@example.com"` with your actual email address.
4. To verify your configuration, type:

   ```
   git config --global --list
   ```

   This will display your current Git configuration settings.

3 Creating a Personal Access Token

A personal access token (PAT) is required to authenticate your Git operations when using HTTPS instead of SSH, especially after GitHub deprecated password-based authentication in 2021. GitHub may update the token creation process in the future, so users should refer to the latest GitHub documentation if the steps differ.

1. Go to your GitHub account settings: https://github.com/settings/tokens.
2. Click on **Generate new token** (classic).
3. Give your token a descriptive name in the **Note** field.
4. Select the scopes or permissions you'd like to grant this token. For general use, **repo** is a common choice.
5. Click **Generate token** at the bottom of the page.
6. Copy the generated token and store it securely, as you will not be able to view it again.
7. Use this token instead of your password when performing Git operations that require authentication:

   ```
   git push https://github.com/username/repo.git
   ```

   Git will prompt you for a username and password. Use your GitHub username and the generated personal access token as the password.

4 Cloning a Repository

1. Navigate to the repository you want to clone.
2. Click on the **Code** button and copy the URL.
3. Open a terminal window on your local machine and navigate to the directory where you want to clone the repository.

   ```
   cd path/to/your/directory
   ```

4. To clone the repository, type:

   ```
   git clone https://github.com/username/repository.git
   ```

5 Forking a Repository

1. Navigate to the repository you want to fork on GitHub.
2. Click the **Fork** button at the top right of the page.
3. This will create a copy of the repository under your own GitHub account.

6 Creating a New Repository

6.1 Initializing a Repository Locally

1. Navigate to the directory where your project is located:

   ```
   cd /path/to/projectfolder
   ```

2. Initialize the directory as a Git repository:

   ```
   git init
   ```

3. Add all files in the directory to the repository:

   ```
   git add .
   ```

4. (Optional) Check the status of your repository:

   ```
   git status -s
   ```

5. Commit the files with an initial commit message:

   ```
   git commit -m "Initial commit"
   ```

6.2 Uploading to GitHub

1. On the GitHub website, create a new repository by clicking the **New** button on your profile page.
2. Name your repository and click **Create repository**.
3. Copy the repository URL (e.g., `https://github.com/username/repo.git`).
4. Link your local repository to the one on GitHub:

   ```
   git remote add origin https://github.com/username/repo.git
   ```

5. Rename the default branch to `main`:

   ```
   git branch -M main
   ```

6. Push your local commits to the GitHub repository:

   ```
   git push -u origin main
   ```

7 Adding and Committing Changes

1. After making changes to files in your repository, you need to stage and commit them. The `git add` command stages changes (adding files to be tracked or marking modified files) for the next commit.
   `git commit` saves those staged changes as a snapshot in the repository's history.
   `git push` uploads these commits from your local repository to a remote repository, such as GitHub.

2. To add all changed files, type:

   ```
   git add .
   ```

   This stages all the modified, new, or deleted files in the current directory and its subdirectories.

3. To add a specific file, type:

   ```
   git add filename.ext
   ```

   Replace `filename.ext` with the name of the file you want to stage.

4. To add multiple specific files, type:

   ```
   git add file1.ext file2.ext
   ```

   Replace `file1.ext` and `file2.ext` with the names of the files you want to stage.

5. If there are files you want Git to ignore (e.g., temporary files, build artifacts), create a `.gitignore` file in the root of your repository:

   ```
   touch .gitignore
   ```

   Open the `.gitignore` file in a text editor and list the files or directories you want to ignore. For example:

   ```
   *.log
   build/
   .DS_Store
   ```

   Note: In this example, `*.log` tells Git to ignore all files with a `.log` extension, `build/` tells Git to ignore the entire `build` directory and all its contents, and `.DS_Store` tells Git to ignore any files named `.DS_Store`, which are hidden system files created by macOS.

6. After staging your changes, commit them by typing:

   ```
   git commit -m "Your commit message here"
   ```

   Replace `"Your commit message here"` with a brief description of the changes you made.

8 Pushing Changes to GitHub

1. To push your committed changes to GitHub, type:

   ```
   git push origin main
   ```

2. Replace `main` with the name of the branch you are pushing to, if different.

9 Branching and Merging

Branches in Git allow you to work on different features or fixes separately from the main codebase. This section explains how to create a branch, switch between branches, and merge branches.

9.1 Creating a New Branch

To create a new branch and switch to it immediately, use:

```
git checkout -b new-branch-name
```

Replace `new-branch-name` with your branch name.

9.2 Switching Between Branches

To switch to an existing branch, use:

```
git checkout branch-name
```

Replace `branch-name` with the name of the branch you want to switch to.

9.3 Listing Branches

To see a list of all branches in your repository, use:

```
git branch
```

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

9.4 Merging Branches

Once you've made changes in a branch and want to incorporate them into another branch (usually `main`), you can merge the branches.

1. First, switch to the branch you want to merge into, typically `main`:

   ```
   git checkout main
   ```

2. Then, merge the changes from your feature branch:

   ```
   git merge new-branch-name
   ```

   Replace `new-branch-name` with the name of the branch you want to merge.

9.5 Resolving Merge Conflicts

Sometimes, Git can't automatically merge changes because the same part of a file was edited in both branches. This creates a merge conflict.

1. Git will mark the conflicting files. Open these files in a text editor to see the conflicts.
2. Conflicting sections will be marked like this:

   ```
   <<<<<<< HEAD
   Your changes
   =======
   Changes in the other branch
   >>>>>>> branch-name
   ```

3. Edit the file to resolve the conflict by choosing which changes to keep, or by combining them.
4. After resolving the conflicts, stage the changes:

   ```
   git add filename.ext
   ```

   Replace `filename.ext` with the name of the file where you resolved conflicts.
5. Finally, commit the merge:

   ```
   git commit
   ```

Note: It's a good practice to regularly pull changes from the remote repository and merge them into your branch to minimize conflicts.

10 Stashing Changes

If you need to switch branches or pull updates but have uncommitted changes you want to save temporarily, you can use Git's stash feature.

10.1 Stashing Your Changes

To stash your changes, type:

```
git stash
```

This saves your changes and reverts your working directory to match the last commit.

10.2 Applying Stashed Changes

To apply your stashed changes back to your working directory, type:

```
git stash apply
```

If you have multiple stashes, you can list them with:

```
git stash list
```

And apply a specific stash with:

```
git stash apply stash@{index}
```

Replace `index` with the number of the stash you want to apply.

11 Viewing Commit History

To see the history of commits in your repository, you can use the following command:

```
git log
```

This will display a list of commits, including the commit hash, author, date, and commit message. You can use various options with `git log` to customize the output, such as:

- `git log --oneline`: Displays a condensed view with just the commit hash and message.
- `git log --graph`: Shows a graphical representation of your branch structure.
- `git log -p`: Shows the diff (changes) introduced in each commit.

12 Reverting Changes

Mistakes happen. If you need to undo changes, Git provides several options depending on what you want to revert.

12.1 Undoing Unstaged Changes

To discard changes in your working directory that haven't been staged, use:

```
git checkout -- filename.ext
```

This reverts the file back to its last committed state.

12.2 Unstaging Files

If you've staged changes but want to unstage them, use:

```
git reset HEAD filename.ext
```

This unstages the file but keeps your changes in the working directory.

12.3 Reverting a Commit

If you've committed changes and need to undo that commit, use:

```
git revert commit-hash
```

Replace `commit-hash` with the hash of the commit you want to revert. This creates a new commit that undoes the changes from the specified commit.

13 Common Git Commands

Here are some commonly used Git commands:

Command	Description
git status	Check the status of your working directory
git add	Stage files for commit
git commit	Commit staged files to the repository
git push	Push commits to a remote repository
git pull	Fetch and merge changes from a remote repository
git clone	Clone a repository to your local machine
git branch	List, create, or delete branches
git checkout	Switch between branches

14 References

GitHub Documentation: https://docs.github.com/en
Pro Git Book: https://git-scm.com/book/en/v2
Git Cheat Sheet: https://education.github.com/git-cheat-sheet-education.pdf
Atlassian Git Tutorials: https://www.atlassian.com/git/tutorials