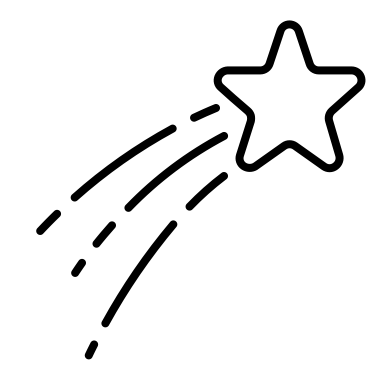
**Introduction to Git and GitHub**

|  |  |
| --- | --- |
| **Overview** | |
| Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.  Git is easy to learn and has a tiny footprint with lightning fast performance. To learn more about Git, refer to [git--fast-version-control](https://git-scm.com/).  For this lab, you assume yourself as a documentation persona. You are responsible for product guides at your organization. There's a release next month, so you need to update a product guide in preparation for the release.  The skills you practice are applicable to any role involved with storing work in Git (code, documentation, any type of file). | |
| **What you'll learn**  At the end of this lab, you will be able to:   * Create a repo * Make changes to the code base * Open up a pull request (PR) * Merge code into the master branch * Create README & gitignore files * Inviting collaborators to your repo | |
| **Setup and requirements** | |
| **Prerequisites**  You will need a GitHub account (it is free). Go to the [GitHub website](https://github.com/join) and set up a GitHub account if you do not have one already. | |
| **Install Git** | |
| Start by installing Git on your instance. | |
| 1 | Download the latest version from their official website:  https://git-scm.com/downloads |
| 2 | Install Git by following the instructions on the official website. |
| 3 | Open Command Prompt (Windows)/ Terminal (MacOS) to verify your installation with the following command: |
| git --version |
| The response will be the git version. If your response is something like -bash: git: command not found, there was a problem with the installation and you should try again. |
| **Task 1. Create a local Git repository** | |
| Now that Git is installed, create a new repository (or repo for short) to hold your release notes. When you work on your own projects, you can create repositories in any directory you want. | |
| 1 | Create a directory for your project. For this lab create a new folder named “gitlab” in your desktop. |
| 2 | Then open the directory with the cd (change directory) command: |
| cd desktop/gitlab |
| 3 | Now, initialize your new Git repository in the folder with the git init command: |
| git init |
| Congratulations! You just initialized a Git repository. | |
| **Task 2. Add a file to the repo** | |
| Your repository is empty. It's time to put something in it! | |
| 1 | Create a new file to the project. You can use any text editor you like when you are working on your own projects. In this lab, you just need to create a txt file using touch command. |
|  | touch <file name>.txt |
| 2 | Check to see which files Git knows about with the git status command: |
| git status |
| What do you see? Git's saying that it knows the file exists, but since you haven't added the file to the repo yet, Git isn't saving it. |
| **Task 3. Add a file to staging** | |
| The next step is to add your file to the staging environment (sometimes referred to as the index). You must do this before you commit a file. When you're working, use the staging environment as just that - a staging ground - for actively editing your files. | |
| 1 | Add the file to the staging environment with the git add command: |
| git add <file name>.txt |
| 2 | Run the git status command again: |
| git status |
| Notice the "Changes to be committed" line. You will see that Git added your file to the staging environment, but the file has not yet been added to a commit. |
| **Task 4. Create a commit** | |
| A commit is a record of what files you have changed since the last commit. Commits allow you to go back to the state of the project at any point in history. You create a commit in order to add files to the Master.  First you need to **identify yourself**, then you can create the commit. | |
| 1 | Run the following, replacing any email id with "you@example.com": |
| git config --global user.email "<you@example.com> |
| 2 | Run the following, replacing any name with "Your Name": |
| git config --global user.name "<Your Name>" |
| 3 | Run the following git commit command. Your message should relate to what's in your commit - for posterity: |
| git commit -m "<your message about the commit>" |
| **Task 5. Create a new branch** | |
| Now that you have mastered the basics, time for something a little more advanced.  Back to your documentation use case, say there is a release coming out next month, and the release includes significant changes to one of your products. You need to update the product guide to help your customers use the new features. You want to make changes to the product guide but you are worried about editing a guide that is actively used for reference by your team and shared with your customers. The solution? **Git branches**.  Branches let you move back and forth between versions, or states, of any given project. In this example, you can create a new branch for the upcoming release without affecting the current product guide. Once you're done with the product guide updates, you can merge your changes from the release branch to the master branch.  **Bonus:** Git tracks the history of branches for you so you always know the history behind all your files and versions. | |
| 1 | Since you're on the master branch already, run the git checkout -b command and name your branch: |
| git checkout -b <branch name> |
| The command will:   * Automatically create a new branch, using the name you specify * Immediately check the branch out to you * Move you to the new branch |
| 2 | Confirm that your branch was created: |
| git status |
| 3 | Switch back to the master branch: |
| git checkout main |
| 4 | Create another file and add it: |
| touch <file name>.txt  git add <file name>.txt |
| 5 | Use git commit -m to stage your changes: |
| git commit -m "<Your message about the commit>" |
| 6 | Return to your branch: |
| git checkout <branch name> |
| 7 | Return back to the master branch: |
| git checkout main |
| Do you see any of the latest commits?  You won't see the changes until you merge them into your new branch. |
| **Task 6. Create a GitHub repo** | |
| If you're the sole owner of your project, you don't need to use GitHub. But if you're on a team, you can use GitHub to collaborate without stepping on each other’s toes. | |
| 1 | To create a new repo on GitHub, go to github.com and log in. |
| 2 | In the left pane, in the Repository section, click New.  GitHub will ask you for some basic info about your repo. |
| 3 | Fill out the form, then click Create repository. |
| GitHub will ask you if you want to start from scratch or add a local project. Since you just created a new repo, you want to push that to GitHub. |
| 4 | Follow the "...or push an existing repository from the command line" section and then copy the GitHub repo URL. |
| 5 | Return to your cmd/terminal window and run the commands: |
| git remote add origin <your GitHub repo URL>  git branch -M main  git push -u origin main |
| **Note:** You will be prompted to log in. Use your GitHub credentials. When prompted for a password, use a personal access token.   * The first command adds the local repository to your GitHub repository. * The second command renames the master (default) branch to ‘main’. * The third command pushes your local repo to GitHub.   Creating a GitHub personal access token.   1. In the upper-right corner of any page, click your profile photo, then click Settings. 2. In the left sidebar, click Developer settings. 3. In the left sidebar, click Personal access tokens. 4. Click Generate new token. 5. Give your token a descriptive name. 6. Select the scopes, or permissions, you’d like to grant this token. To use your token to access repositories from the command line, select repo. 7. Click Generate token. 8. Copy the token to your clipboard. For security reasons, after you navigate off the page, you will not be able to see the token again. |
| **Task 7. Push a branch to GitHub** | |
| Pushing the commit in your branch to your new GitHub repo allows other people to see the changes you've made (think code review). The repository's owner can review changes prior to merging to the master branch. | |
| 1 | Run the following command to push changes onto a new branch on GitHub, replacing branch name with a name of your branch: |
| git push origin <your branch name> |
| Does GitHub automatically create the branch for you on the remote repository? |
| 2 | In GitHub, click on the Code tab. You should now see the branch name you just pushed. |
| 3 | Click the Compare & pull request button that is next to your branch name. |
| You'll now see the name of your commit. In a production environment you could leave a comment about this commit. |
| 4 | For this lab click Create pull request. |
| GitHub verifies that the files you are adding aren't in conflict with the Master copy. If everything checks out, and it should since these are new files, you will see a green check mark. |
| 5 | Click Merge pull request, then Confirm merge. |
| You'll see a "Pull request successfully merged and closed" message. |
| 6 | At this point you can click the Delete branch to clean up. You don't have to do this, but you may end up with a mess if you have too many branches. Notice that you'll have another chance to leave comments. |
| **Task 8. Update a file** | |
| You have added new files to your repo. Now go back to your cmd/terminal window to edit one of those files, then commit that file to GitHub. This will show you how GitHub manages changes. | |
| 1 | Back in your cmd/terminal window, switch to the Master branch and review the files it has: |
| git checkout main |
| 2 | Now create and name a new branch for the work you're about to do: |
| git checkout -b <new branch name> |
| 3 | Edit one of your files. Type some content into the .txt file, then save it. |
| 4 | Add the file to the staging environment: |
| git add <file name>.txt |
| 5 | Now commit this change to GitHub: |
| git commit -m <new comment> |
| 6 | Push this to the new branch: |
| git push origin <new branch name> |
| **Task 9. Create a Pull Request** | |
| Now that you have added your updates, you need to notify a repo's owner that you want to make changes to their files through a pull request. A pull request (PR) allows them to review your changes to make sure it looks good before putting your changes on the master branch (which are documentation updates in this use case, but could also be code changes or any number of other cases).  Since you are the sole owner of your repo for this lab, you don't have to create a PR to merge your changes, but it's a good idea to create one anyway to track the history of your updates. | |
| Create your PR in GitHub | |
| 1 | Return to your GitHub window. |
| 2 | Open a pull request by clicking the **Pull requests** tab, then **New pull request**. |
| 3 | In the Compare and review section, click on your branch name.  You'll be shown what changed in the file that you're adding to Master. |
| 4 | Click Create pull request.  You'll see the name of the second commit you created. |
| 5 | Now click Create pull request.  GitHub verifies that your changes are good. |
| 6 | Click the Merge pull request to add your changes into the master branch. |
| 7 | Click Confirm merge. |
| Note: Is the button gray? Probably not right now. When this button is gray, you have a merge conflict. A change in one file conflicts with a change in another, and Git can't decide which one is correct. If you have a merge conflict, you have to tell Git which version to use. |
| **Task 10. Clean up and verification** | |
| 1 | Clean up after yourself by deleting your branch, click the Delete branch button. |
| 2 | Double check that your commits were merged by clicking the Commits link on the first page of your new repo. |
| 3 | Find your commit. |
| 4 | Notice the hash code of the commit. |
| Note: It's worth mentioning that the hash code is unique for the specific commit. It's useful for identifying specific commits when you're reviewing (and maybe undoing) changes with the Git revert command:  git revert  You don't need to run this command now. |
| **Task 11. Sync your local project with GitHub** | |
| Right now, the commit that you made in your branch then merged into the master branch doesn't exist in the version of Master on your local machine. Time to update your local version of Master. | |
| 1 | Use the git pull origin master command (specific to the master branch) to get the most recent changes from GitHub to your local repo: |
| git pull origin main |
| 2 | Use the git log command to see all new commits: |
| git log  press q to exit log |
| 3 | Switch back to the master branch in your remote session. Use the git checkout master command: |
| git checkout main |
| 4 | If you see the warning "Your branch is behind....", this means your local branch needs to add the commits from GitHub Master, so run the following: |
| git pull origin main |
| 5 | Then run the following to confirm everything is synced: |
| git checkout main |
| 6 | You should see a message that confirms your local repository is now up to date. |
| **Task 12. Create README file** | |
| You can add a README file to your repository to tell other people why your project is useful, what they can do with your project, and how they can use it.  A README is often the first item a visitor will see when visiting your repository. README files typically include information on:   * What the project does * Why the project is useful * How users can get started with the project * Where users can get help with your project * Who maintains and contributes to the project | |
| 1 | Create a new README file |
| touch <file name>.md |
| 2 | Edit the README file using any editor. |
| 3 | Add the file to the staging environment: |
| git add <file name>.md |
| 4 | Commit the file |
| git commit -m “adding a README file” |
| 5 | Push the file to GitHub |
| git push origin main |
| **Task 13. Create gitignore file** | |
| You can create a *.gitignore* file in your repository's root directory to tell Git which files and directories to ignore when you make a commit. To share the ignore rules with other users who clone the repository, commit the *.gitignore* file in to your repository. | |
| 1 | Create a new gitignore file |
| touch <filename>.gitignore |
| 2 | Using any editor, edit your gitignore file by referring to some common .gitignore configurations.  <https://gist.github.com/octocat/9257657>  You can also refer to some of the .gitignore templates here:  <https://github.com/github/gitignore> |
| 3 | Add the file to the staging environment: |
| git add <file name>.gitignore |
| 4 | Commit the file |
| git commit -m “adding a gitignore file” |
| 5 | Push the file to GitHub |
| git push origin main |
| **Task 14. Inviting collaborators to a personal repository** | |
| You can invite users to become collaborators to your personal repository. | |
| 1 | Ask for the username of the person you're inviting as a collaborator. |
| 2 | On GitHub.com, navigate to the main page of the repository. |
| 3 | Under your repository name, click **Settings**. |
| 4 | In the "Access" section of the sidebar, click Collaborators & teams. |
| 5 | Click **Invite a collaborator**. |
| 6 | In the search field, start typing the name of person you want to invite, then click a name in the list of matches. |
| 7 | Click **Add NAME to REPOSITORY**. |
| 8 | The user will receive an email inviting them to the repository. Once they accept your invitation, they will have collaborator access to your repository. |

Congratulations!!! You have completed the workshop successfully.