Getting Started with Git and GitHub

Table of Contents

[MODULE 1 Git and GitHub Fundamentals 1](#_Toc180777916)

[Summary: Getting Started with Git and GitHub 2](#_Toc180777917)

[Summary: Branches with GitHub 2](#_Toc180777918)

[Module 1 Glossary: Git and GitHub Fundamentals 2](#_Toc180777919)

[MODULE 2 Git Commands and Managing GitHub Projects 3](#_Toc180777920)

[Reading: Git Commands 3](#_Toc180777921)

[Example 1 6](#_Toc180777922)

[Example 2 6](#_Toc180777923)

[Summary: Git Workflows with Git Commands 7](#_Toc180777924)

[Module 2 Cheat Sheet: Git Commands and Managing GitHub Projects 8](#_Toc180777925)

[Module 2 Glossary: Git Commands and Managing GitHub Projects 8](#_Toc180777926)

[MODULE 3 Final Project Assessment 9](#_Toc180777927)

[Course Glossary: Getting Started with Git and GitHub 9](#_Toc180777928)

[MODULE 4 Working with GitHub from your Windows Desktop 10](#_Toc180777929)

## **MODULE 1 Git and GitHub Fundamentals**

## Summary: Getting Started with Git and GitHub

* Git is a versatile version control system used for tracking changes in code and collaborating with others on software projects.
* Due to a distributed version control system, Git enables you to revert to the previous state or review the project’s history.
* GitHub is one of the most popular web-hosted services for Git repositories.
* Repositories are storage structures that store documents, including application source code, and enable contributors to track and maintain version control.
* Git repository model
* Primarily focused on tracking source code during development.
* Contains elements to coordinate among programmers, track changes, and support non-linear workflows.
* Repositories are storage structures that can hold Code, track Issues, and enable you to collaborate with others.
* GitHub enables you to create repositories, edit files using the web interface, commit the changes to the file, upload the files, and a lot more.

## Summary: Branches with GitHub

* A branch is a snapshot of your repository to which you can make changes.
* In the child branch, you can build, make edits, test the changes, and then merge them with the main branch.
* To ensure that changes are made by one member, do not impede or affect the workflow of other members, multiple branches can be created and merged with the main branch.
* A pull request is a way to notify other team members of the changes and edits made to the main branch.

## Module 1 Glossary: Git and GitHub Fundamentals

|  |  |
| --- | --- |
| Term | Definition |
| Branch | A separate line of development that allows to work on features or fixes independently. |
| Clone | A local copy of the remote Git repository on the computer. |
| Commit | A snapshot of the project's current state at a specific point in time, along with a description of the changes made. |
| Continuous delivery (CD) | The automated movement of software through the software development lifecycle. |
| Continuous integration (CI) | A software development process in which developers integrate new code into the code base at least once a day. |
| Distributed version control system (DVCS) | A system that keeps track of changes to code, regardless of where it is stored. Multiple users work on the same codebase or repository, mirroring the codebase on their computers if needed, while the distributed version control software helps manage synchronization amongst the various codebase mirrors. |
| Fork | A copy of a repository into your GitHub account. |
| GitHub | A web-hosted service for the Git repository. |
| GitHub branches | A branch stores all files in GitHub. Branches are used to isolate changes to code. When the changes are complete, they can be merged back into the main branch. |
| GitLab | A complete DevOps platform delivered as a single application. It provides access to Git repositories, controlled by source code management. |
| Git | Free and open-source software distributed under the GNU General Public License. It is a distributed version control system that allows users to have a copy of their own project on their computer anywhere in the world. |
| Merge | A process to combine changes from one branch to another, typically merging a feature branch into the main branch. |
| Pull request | A process used to request that someone review and approve your changes before they become final. |
| Repository | A data structure for storing documents, including application source code. It contains the project folders that are set up for version control. |
| SSH Protocol | A method for secure remote login from one computer to another. |
| Version control | A system that allows you to keep track of changes to your documents. This process allows you to recover older versions of the documents if any mistakes are made. |
| Working directory | A directory in your file system that contains files and subdirectories on your computer that are associated with a Git repository. |

Top of Form

Bottom of Form

## **MODULE 2 Git Commands and Managing GitHub Projects**

## Reading: Git Commands

In this reading, you will summarize and describe additional Git commands that you may use while working on your projects. You will also look at the syntax for each command.

Git is a widely used version control system that offers numerous benefits to developers and teams working on software development projects.

Let's look at some useful Git commands and understand them:

1. **git add**
   * *Description*: It adds changes to the staging area. This command stages the changes made to the files and prepares them for the next commit.
   * *Syntax*:
     + **git add <filename.txt>** (to add a specific file)
     + **git add .** (to add all the files that are new or changed in the current directory)
     + **git add -A** (to add all changes in the entire working tree, from the root of the repository, regardless of where you are in the directory structure)
2. **git reset**
   * *Description*: It resets changes in the working directory. When used with –hard HEAD, this command discards all changes made to the working directory and staging area and resets the repository to the last commit (HEAD).
   * *Syntax*:
     + **git reset**
     + **git reset –hard HEAD**
3. **git branch**
   * *Description*: It lists, creates, or deletes branches in a repository. To delete the branch, first check out the branch using **git checkout** and then run the command to delete the branch locally.
   * *Syntax*:
     + **git branch** (to list branches)
     + **git branch <new-branch>** (to create a new branch)
     + **git branch -d <branch-name>** (to delete a branch)
4. **git checkout main**
   * *Description*: It switches to the "main" branch. This will switch your current branch to "main."
   * *Syntax*: **git checkout main**
5. **git clone**
   * *Description*: It copies a repository from a remote source to your local machine. This will create a copy of the repository in your current working directory.
   * *Syntax*: **git clone <repository URL>**
6. **git pull**
   * *Description*: It fetches changes from a remote repository and merges them into your local branch. First, switch to the branch that you want to merge changes into by running the **git checkout** command. Then, run the **git pull** command, which will fetch the changes from the main branch of the origin remote repository and merge them into your current branch.
   * *Syntax*: **git pull origin main**
7. **git push**
   * *Description*: It uploads local repository content to a remote repository. Make sure you are on the branch that you want to push by running the **git checkout** command first, then push the branch to the remote repository.
   * *Syntax*: **git push origin <branch-name>**
8. **git version**
   * *Description*: It displays the current Git version installed on your system.
   * *Syntax*: **git version**
9. **git diff**
   * *Description*: It shows changes between commits, commit and working tree, etc. It also compares the branches.
   * *Syntax*:
     + **git diff** (shows the difference between the working directory and the last commit)
     + **git diff HEAD~1 HEAD** (shows the difference between the last and second-last commits)
     + **git diff <branch-1> <branch-2>** (compares the specified branches)
10. **git revert**
    * *Description*: It reverts a commit by applying a new commit. This will create a new commit that undoes the changes made by the last commit.
    * *Syntax*: **git revert HEAD**
11. **git config –global user.email <Your GitHub Email>**
    * *Description*: It sets a global email configuration for Git. This needs to be executed before doing a commit to authenticate the user's email ID.
    * *Syntax*: **git config –global user.email <Your GitHub Email>**
12. **git config –global user.name <Your GitHub Username>**
    * *Description*: It sets a global username configuration for Git. This needs to be executed before doing a commit to authenticate users' username.
    * *Syntax*: **git config –global user.name <Your GitHub Username>**
13. **git remote**
    * *Description*: It lists the names of all remote repositories associated with your local repository.
    * *Syntax*: **git remote**
14. **git remote -v**
    * *Description*: It lists all remote repositories that your local Git repository is connected to, along with the URLs associated with those remote repositories.
    * *Syntax*: **git remote -v**
15. **git remote add origin <URL>**
    * *Description*: It adds a remote repository named "origin" with the specified URL.
    * *Syntax*: **git remote add origin <URL>**
16. **git remote rename**
    * *Description*: The git remote rename command is followed by the name of the remote repository (origin) you want to rename and the new name (upstream) you want to give it. This will rename the "origin" remote repository to "upstream."
    * *Syntax*: **git remote rename origin upstream**
17. **git remote rm <name>**
    * *Description*: It adds a remote repository named "origin" with the specified URL.
    * *Syntax*: **git remote** *rm origin*
18. **git format-patch**
    * *Description*: It generates patches for email submission. These patches can be used for submitting changes via email or for sharing them with others.
    * *Syntax*: **git format-patch HEAD~3** (creates patches for the last three commits)
19. **git request-pull**
    * *Description*: It generates a summary of pending changes for an email request. It helps communicate the changes made in a branch or fork to the upstream repository maintainer.
    * *Syntax*: **git request-pull origin/main <myfork or branch\_name>**
20. **git send-email**
    * *Description*: It sends a collection of patches as emails. It allows you to send multiple patch files to recipients via email. Please make sure to set the email address and name using the **git config** command so that the email client knows the sender's information when sending the emails.
    * *Syntax*: **git send-email \*.patch**
21. **git am**
    * *Description*: It applies patches to the repository. It takes a patch file as input and applies the changes specified in the patch file to the repository.
    * *Syntax*: **git am <patchfile.patch>**
22. **git daemon**
    * *Description*: It exposes repositories via the Git:// protocol. The Git protocol is a lightweight protocol designed for efficient communication between Git clients and servers.
    * *Syntax*: **git daemon –base-path=/path/to/repositories**
23. **git instaweb**
    * *Description*: It instantly launches a web server to browse repositories. It provides a simplified way to view repository contents through a web interface without the need for configuring a full web server.
    * *Syntax*: **git instaweb –httpd=webrick**
24. **git rerere**
    * *Description*: It reuses recorded resolution of previously resolved merge conflicts. Please note that rerere.enabled configuration option needs to be set to "true" (**git config –global rerere.enabled true**) for git rerere to work. Additionally, note that git rerere only applies to conflicts that have been resolved using the same branch and commit.
    * *Syntax*: **git rerere**
25. **git init**
    * *Description*: Creates a new local repository
    * *Syntax*: **git init**

## Example 1

* git clone https://github.com/mrdanwa/coursera.git
* cd coursera
* git branch
* git branch child
* git branch
* git checkout child
* ls
* testchild.py
* nano testchild.py
* git status
* git add testchild.py
* git commit -m "Made some changes"
* git checkout main
* git merge child
* git push -u origin main

## Example 2

* ORIGIN=https://github.com/mrdanwa/gkpbt-css-circle.gittheia@theia-dwang1:/home/project$
* ls
* git clone $ORIGIN
* ls
* cd gkpbt-css-circle
* git checkout -b feature-circle-500
* git branch
* git status
* git diff ./style.css
* git add .
* git status
* git config --global user.name "mrdanwa"
* git config --global user.email "wdanrui@gmail.com"
* git commit -m "Changing the height and the width of the circle"
* git status
* git checkout main
* git merge feature-circle-500
* git log
* git log --oneline
* git checkout main
* git branch -d feature-circle-500
* git branch
* git push -u origin main

## Summary: Git Workflows with Git Commands

* GitHub has over 100 million repositories. You can clone a repository and sync changes back to the original. You can also fork a repository and use it as the base for the new project or work on that project independently.
* The steps included in a GitHub workflow are:
  + Clone the remote repository or initialize a Git repository.
  + Move files to a staging area.
  + Perform an initial commit.
  + Create a branch and work on it.
  + Add files to the staging area and commit.
  + Push local commits to the remote repository.
  + Create a pull request for review and merging.
  + Use the pull operation to update the local repository.
* Multiple roles are involved in managing a project: Developer, Integrator, and Repository Administrator.
  + A Developer working in a group project uses commands like git clone, git pull, git fetch, git push, and git request-pull in addition to the ones needed by a standalone developer.
  + An Integrator in a group project reviews and integrates changes made by others. Integrators use commands like git pull, git revert, and git push in addition to the ones needed by participants.
  + Repository Administrators structure how the repository is organized and how users interact with the repository. They also configure the servers needed for accessing the web services and documentation, define email and index settings, and manage the look and feel of the application.
* The following table shows various Git commands:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| git init | git checkout | git revert | git-format-patch | git fetch upstream |
| git status | git merge | git config --global user.email | git-request-pull | git merge upstream/main |
| git add . | git clone | git config --global user.name | git-send-email | git pull upstream |
| git commit | git pull | git remote -v | git-am | git web |
| git log | git push | git remote rename | git-daemon | git-instaweb |
| git reset | git version | git remote add origin | git remote -v | git-pull downstream |
| git branch | git diff | git-remote | git remote add upstream | git-rerere |

## Module 2 Cheat Sheet: Git Commands and Managing GitHub Projects

<https://author-ide.skills.network/render?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJtZF9pbnN0cnVjdGlvbnNfdXJsIjoiaHR0cHM6Ly9jZi1jb3Vyc2VzLWRhdGEuczMudXMuY2xvdWQtb2JqZWN0LXN0b3JhZ2UuYXBwZG9tYWluLmNsb3VkL0lCTVNraWxsc05ldHdvcmstQ0QwMTMxRU4tQ291cnNlcmEvbGFicy9DaGVhdF9zaGVldC9NMi9Vc2luZ19HaXRfQ29tbWFuZHNfYW5kX01hbmFnaW5nX0dpdEh1Yl9Qcm9qZWMubWQiLCJ0b29sX3R5cGUiOiJpbnN0cnVjdGlvbmFsLWxhYiIsImFkbWluIjpmYWxzZSwiaWF0IjoxNzAyMjcxMTY5fQ.Q9GDgatZl3_dvcpuDMnNXMpWo_h3hSt9gZL-1LtaAW4>

## Module 2 Glossary: Git Commands and Managing GitHub Projects

|  |  |
| --- | --- |
| Term | Definition |
| Cloning | A process of creating a copy of the project's code and its complete version history from the remote repository on the local machine. |
| Commit | A snapshot of the project's current state at a specific point in time, along with a description of the changes made. |
| Developer | A computer programmer who is responsible for writing code. |
| Distributed version control system (DVCS) | A system that keeps track of changes to code, regardless of where it is stored. Multiple users work on the same codebase or repository, mirroring the codebase on their computers if needed, while the distributed version control software helps manage synchronization amongst the various codebase mirrors. |
| Fork | A copy of a repository into your GitHub account. |
| Forking | Forking creates a copy of a repository on which one can work without affecting the original repository. |
| GitHub | A web-hosted service for the Git repository. |
| Git | A free and open-source software distributed under the GNU General Public License. It is a distributed version control system that allows users to have a copy of their own project on their computer anywhere in the world. |
| Integrator | A role that is responsible for managing changes made by developers. |
| Master branch | A branch that stores the deployable version of your code. The master branch is created by default and is definitive. |
| Merge | A process to combine changes from one branch to another, typically merging a feature branch into the main branch. |
| Origin | A term that refers to the repository where a copy is cloned from. |
| Pull request | A process used to request that someone review and approve your changes before they become final. |
| Remote repositories | Repositories that are stored elsewhere. They can exist on the internet, on your network, or on your local computer. |
| Repository administrator | A role that is responsible for configuring and maintaining access to the repository. |
| Repository | A data structure for storing documents, including application source code. It contains the project folders that are set up for version control. |
| Staging area | An area where commits can be formatted and reviewed before completing the commit. |
| Upstream | A term used by developers to refer to the original source where the local copy was cloned from. |
| Version control | A system that allows you to keep track of changes to your documents. This process allows you to recover older versions of the documents if any mistakes are made. |

## **MODULE 3 Final Project Assessment**

## Course Glossary: Getting Started with Git and GitHub

|  |  |
| --- | --- |
| Term | Definition |
| Branch | A separate line of development that allows to work on features or fixes independently. |
| Clone | A local copy of the remote Git repository on the computer. |
| Cloning | A process of creating a copy of the project's code and its complete version history from the remote repository on the local machine. |
| Commit | A snapshot of the project's current state at a specific point in time, along with a description of the changes made. |
| Continuous delivery (CD) | The automated movement of software through the software development lifecycle. |
| Continuous integration (CI) | A software development process in which developers integrate new code into the code base at least once a day. |
| Developer | A computer programmer who is responsible for writing code. |
| Distributed version control system (DVCS) | A system that keeps track of changes to code, regardless of where it is stored. It allows multiple users to work on the same codebase or repository, mirroring the codebase on their computers if needed, while the distributed version control software helps manage synchronization amongst the various codebase mirrors. |
| Fork | A copy of a repository. You can fork a repository to use it as the base for a new project or to work on a project independently. |
| Forking | Forking creates a copy of a repository on which one can work without affecting the original repository. |
| Git | Free and open-source software distributed under the GNU General Public License. It is a distributed version control system that allows users to have a copy of their own project on their computer anywhere in the world. |
| GitHub | A web-hosted service for the Git repository. |
| GitHub branches | A branch stores all files in GitHub. Branches are used to isolate changes to code. When the changes are complete, they can be merged back into the main branch. |
| GitLab | A complete DevOps platform delivered as a single application. It provides access to Git repositories, controlled by source code management. |
| Integrator | A role that is responsible for managing changes made by developers. |
| Master branch | A branch that stores the deployable version of your code. The master branch is created by default and is definitive. |
| Merge | A process to combine changes from one branch to another, typically merging a feature branch into the main branch. |
| Origin | A term that refers to the repository where a copy is cloned from. |
| Pull request | A process used to request that someone reviews and approves your changes before they become final. |
| Remote repositories | Repositories that are stored elsewhere. They can exist on the internet, on your network, or on your local computer. |
| Repository | A data structure for storing documents, including application source code. It contains the project folders that are set up for version control. |
| Repository administrator | A role that is responsible for configuring and maintaining access to the repository. |
| SSH protocol | A method for secure remote login from one computer to another. |
| Staging area | An area where commits can be formatted and reviewed before completing the commit. |
| Upstream | A term used by developers to refer to the original source where the local copy was cloned from. |
| Version control | A system that allows you to keep track of changes to your documents. This process allows you to recover older versions of the documents if any mistakes are made. |
| Working directory | A directory in your file system that contains files and subdirectories on your computer that are associated with a Git repository. |

## **MODULE 4 Working with GitHub from your Windows Desktop**