UPES

Xebia

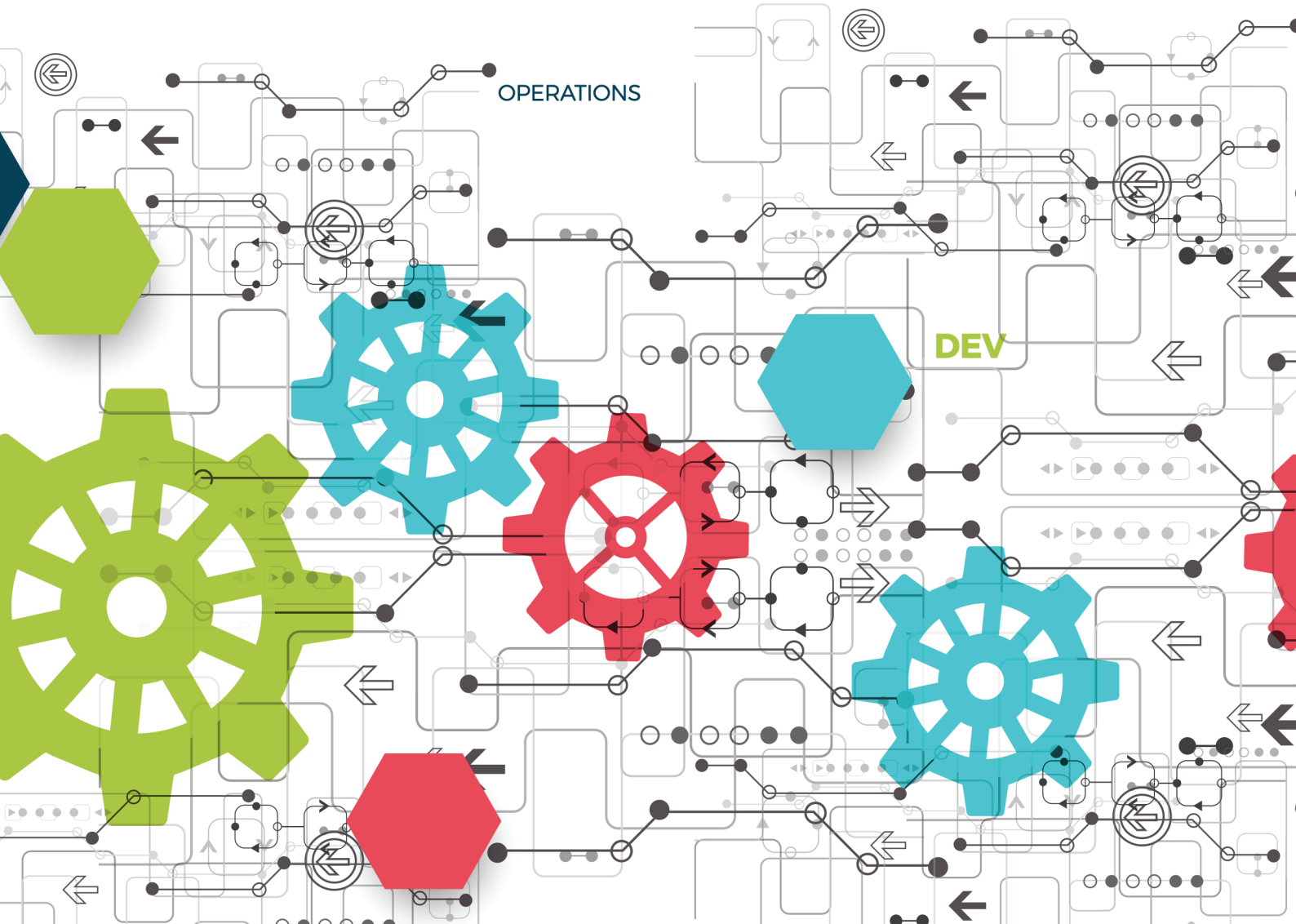
YEAR

B.Tech Computer Science and Engineering in DevOps

Source Code Management

Semester 02 | Lab

Release 1.0.0



Copyright & Disclaimer

B. TECH CSE with Specialization in Source Code Management

Version 1.0.0

Acknowledgements

We would like to sincerely thank the experts who have contributed to and shaped B. TECH CSE with Specialization in DevOps. Version 1.0.0
Semester 02

SME

Rajagopalan Varadan
A tech enthusiast who loves learning and working with cutting-edge technologies like DevOps, Big Data, Data science, Machine Learning, AWS & Open stack

Course Reviewers.

Aditya Kalia | Xebia
Maneet Kaur | Xebia
Sandeep Singh Rawat | Xebia
Abhishek Srivastava | Xebia
Rohit Sharma | Xebia

Review Board Members.

Anand Sahay | Xebia

Xebia Group consists of seven specialized, interlinked companies: Xebia, Xebia Academy, XebiaLabs, StackState, GoDataDriven, Xpirit and Binx.io. With offices in Amsterdam and Hilversum (Netherlands), Paris, Delhi, Bangalore and Boston, we employ over 700 people worldwide. Our solutions address digital strategy; agile transformations; DevOps and continuous delivery; big data and data science; cloud infrastructures; agile software development; quality and test automation; and agile software security.

ODW is dedicated to provide innovative and creative solutions that contribute in growth of emerging technologies. As a learning experience provider, ODW strengths include providing unique, up to date content by combining industry best practices with leading edge technology. ODW delivers high quality solutions and services which focus on digital learning transformation.

Source Code Management Lab

Table of Contents

1. Introducing Version Control System - Git

In this section, we will introduce Git – A distributed version control system. Let us begin by first installing and configuring Git.

1.1 Installing Git CLI

There are several ways to install Git CLI. You can either use a package manager or binary installer or build and install from the source code.

1.1.1. Installing on Linux

The easiest way to install Git CLI is to use the package manager of your Linux distribution.

→ On Debian/Ubuntu

apt-get is a package manager for Debian based Linux distribution.

- To install the latest stable version of Git CLI, run below command.

  $ sudo apt-get install -y git

- To verify the Git installation, run below command.

  $ git --version

```
$ git --version
git version 2.7.4
$
```

→ On RHEL/Centos

yum is a package manager for RHEL/Centos based Linux distribution.

- To install the latest stable version of Git CLI, run below command.

  $ sudo yum install -y git

- To verify the Git installation, run below command.

  $ git --version

```
$ git --version
git version 2.7.4
$
```

→ On any other Linux distribution

Refer - https://git-scm.com/download/linux

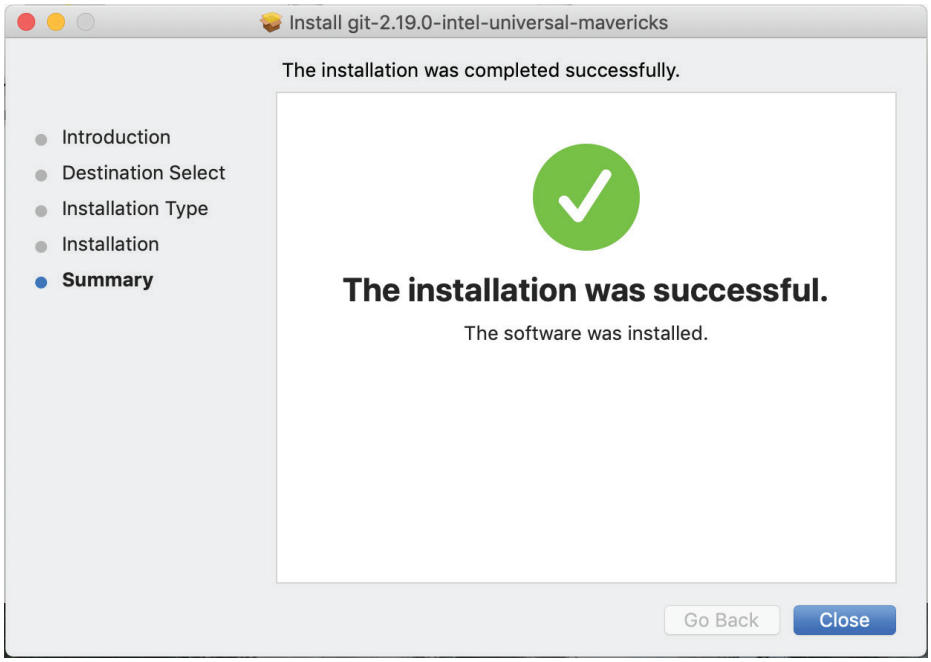
1.1.2. Installing on Mac

The easiest way to install Git CLI on Mac is to use a binary installer.

Steps:

- Download the binary installer from https://git-scm.com/download/mac
- Follow the instructions in the wizard to complete the installation.

Upon successful installation, you will see below screen.



To verify the Git installation, open the terminal and run below command.

$ git --version

```
$ git --version
git version 2.7.4
$
```

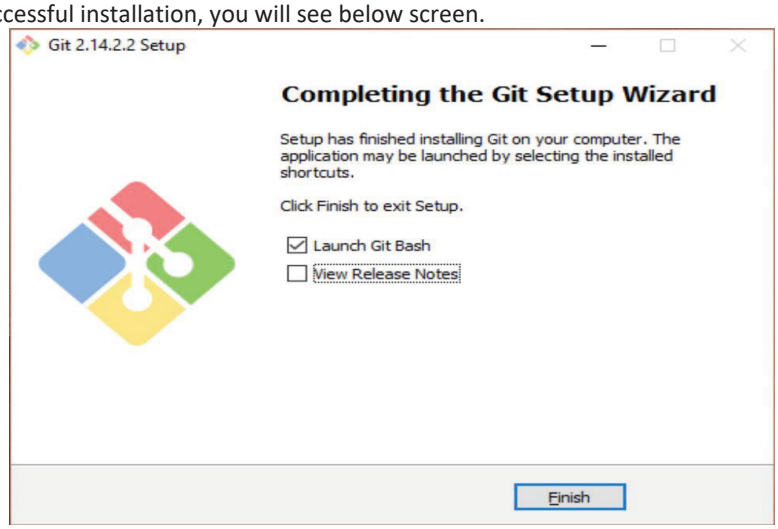
1.1.3. Installing on Windows

The easiest way to install Git CLI on Windows is to use a binary installer.

Steps:

- Download the binary installer from https://git-scm.com/download/win
- Follow the instructions in the wizard to complete the installation.

Upon successful installation, you will see below screen.



To verify the Git installation, open the Git Bash prompt and run below command.

$ git --version

```
$ git --version
git version 2.7.4
$
```

1.2. Configuring Git

Before we begin working with Git repository, let us have the basic configuration in place.

- To list the existing global config, run below command.

  $ git config --global --list

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

For fresh installation of Git, you will have no config set as seen in the above screenshot.

- To tell Git who you are and set your account's default identity, run below command.

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

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

- Omit *--global* to set the identity only in the current repository.

```
$ git config --global user.email "arya.stark@winterfell.com"
$ git config --global user.name "Arya Stark"
$
$ git config --global --list
user.email=arya.stark@winterfell.com
user.name=Arya Stark
$
```


1.3. Installing Git GUI

Git GUI can be used for visualizing the Git operations. There are plenty of third-party GUI clients available for Git. You can find them here - https://git-scm.com/downloads/guis.

Based on your platform, download and follow the instructions in the wizard or installation guide to complete the installation.

1.4. Exploring git --help

git --help command displays the help information about Git. Let us decipher the ***git --help*** command output.

$ git --help

- Section 1: Displays the Git usage pattern
- Section 2: Common Git commands used in various situations
- Section 3: Help on available subcommands

```
$ git --help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status

grow, mark and tweak your common history
   branch     List, create, or delete branches
   checkout   Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   merge      Join two or more development histories together
   rebase     Reapply commits on top of another base tip
   tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch      Download objects and refs from another repository
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
$
```

- ***git help -a*** command lists the available subcommands

  $ git help -a

- ***git help <command>*** opens the Git manual for the subcommand
  E.g.:
  $ git help add

- ***git help -g*** command lists the common Git concept guides

  $ git help -g

- ***git help <concept>*** opens the Git manual for the Git concept
  E.g.:

  $ git help hooks

1.5. Setting up a Git Repository

In this section, we will explore different ways of setting up a Git repository.

1.5.1. Initialize an Empty Git Repository

Here, we will see how to create a repository from scratch

Steps:

- Open the terminal or Git Bash prompt on your system
- Create a new directory for your Git project

  $ mkdir my-first-git-repo

- Change to the newly created project directory

  $ cd my-first-git-repo/

- Type:

  $ git init

Upon successful initialization, a new subdirectory named ***.git*** will be created that will hold the metadata about your project

```
$ git init
Initialized empty Git repository in /home/user/my-first-git-repo/.git/
$
```


1.5.2. Initialize a Repository in an Existing Directory

Here, we will see how to convert an existing project directory that is not currently under version control into a Git repository

Steps:

- Open the terminal or Git Bash prompt on your system
- Move to your existing project directory where you want the version control

  $ cd my-existing-project-directory/

- Type:

  $ git init

Upon successful initialization, a new subdirectory named ***.git*** will be created that will hold the metadata about your project

```
$ git init
Initialized empty Git repository in /home/user/my-existing-project-directory/.git/
```

1.5.3. Clone an Existing Repository

Here, we will see how to get a copy of an existing Git repository

Steps:

- Open the terminal or Git Bash prompt on your system
- The command to clone a repository is ***git clone <URL>***

  E.g.: To clone Python-Sample-Application, get the web URL from GitHub

  $ git clone https://github.com/uber/Python-Sample-Application.git

- This creates a directory named ***Python-Sample-Application,*** initializes a ***.git*** subdirectory and downloads all the data for that repository

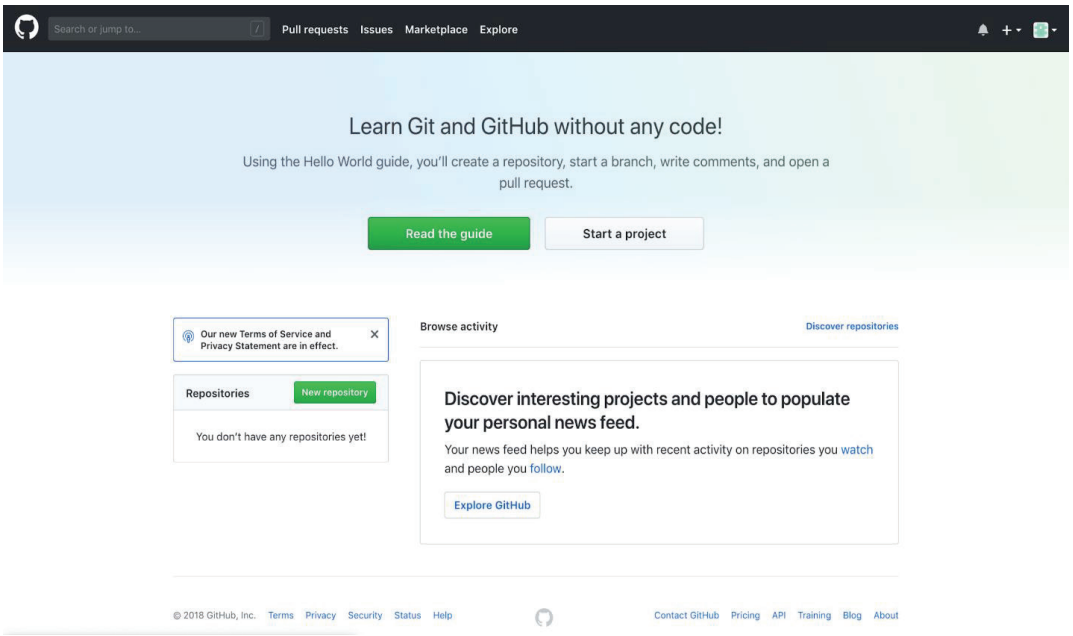
```
$ git clone https://github.com/uber/Python-Sample-Application.git
Cloning into 'Python-Sample-Application'...
remote: Enumerating objects: 224, done.
remote: Total 224 (delta 0), reused 0 (delta 0), pack-reused 224
Receiving objects: 100% (224/224), 42.28 KiB | 0 bytes/s, done.
Resolving deltas: 100% (107/107), done.
Checking connectivity... done.
$
$ ls
Python-Sample-Application
$
```

2. Introducing GitHub

GitHub is a web service to host Git repositories that offers all of the distributed version control and source code management functionality of Git.

2.1. Exploring GitHub

- Open https://github.com/ on your browser
- If you don't already have an account, complete the Sign-up form and create an account
- In a free user account, you can only create public repositories
- After Sign-in/Sign-up, you will see the homepage as below



- As you can see in the above screenshot, there is an option to search repositories, code, commits, issues, topics, wikis, users, etc., in the search bar, view pull requests, issues, available marketplace integration with Git and an option to explore other resources.
- In the upper right corner, the + dropdown provides the option of creating new repository, import repository, create new gist, and new organization.
- The avatar dropdown on the top right corner gives the option to view your profile, repositories, and gists and edit account setting.

2.2. Creating a Public Repository

In this section, we will explore the different ways of creating a public repository. As previously mentioned, in a free user account, you can only create public repositories. To create private repositories, upgrade your user account by providing the payment details.

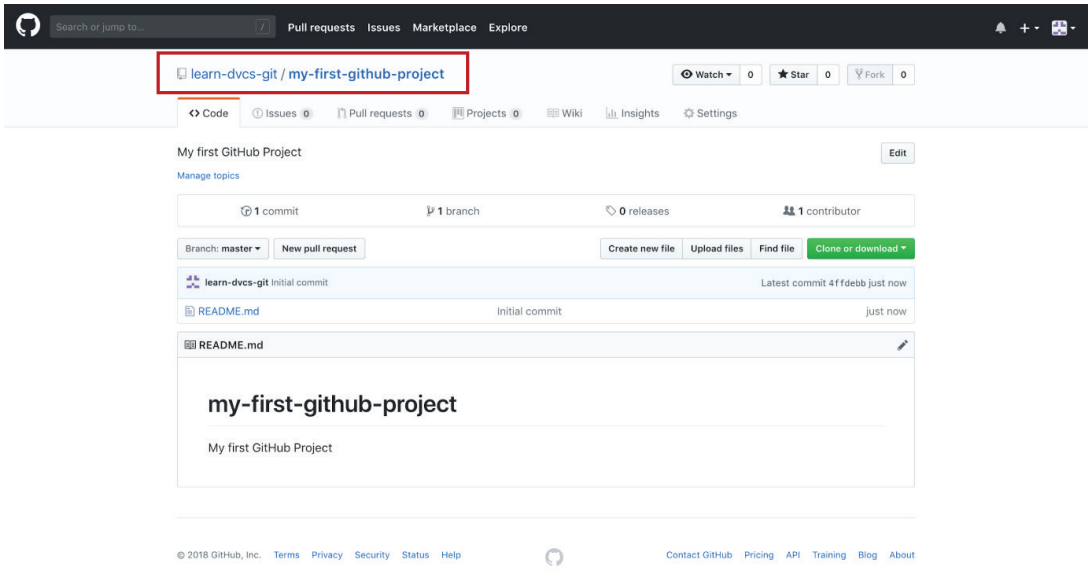
2.2.1. Create an Empty Git Repository

Here, we will see how to create an empty Git repository in GitHub.

Steps:

- Open https://github.com/ on your browser
- In the upper right corner, click on **+** and then select **New repository**
- Enter repository name
- Optionally, write a short project description
- Select **Initialize this repository with a README**
- Click on **Create repository**

A view of repository dashboard:



2.2.2. Fork an Existing Repository

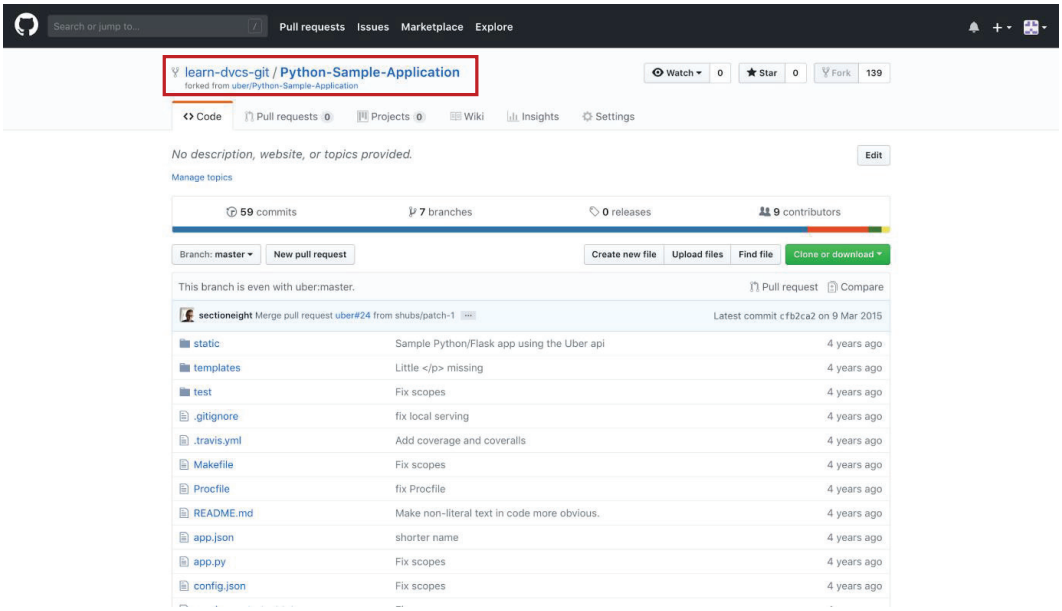
Here, we will see how to fork an existing repository in GitHub. Typically, forking is done when you don't have push access to an existing repository to collaborate. By forking, you get a copy of the repository in your namespace with complete access to it.

Steps:

- Open https://github.com/ on your browser

- Search for the project you want to fork. Let us suppose we decide to fork the Python-Sample-Application (https://github.com/uber/Python-Sample-Application)
- Visit the project page, click on ***Fork*** button on the top right corner

- Once forking is complete, you will be redirected to the forked project page in your namespace.



2.2.3. Push an Existing Repository to GitHub

Here, we will see how we can push existing local repository to a public repository in GitHub.

Steps:

- Open the terminal or Git Bash prompt on your system
- Move to your local repository

  $ cd local-to-public-repo-demo/

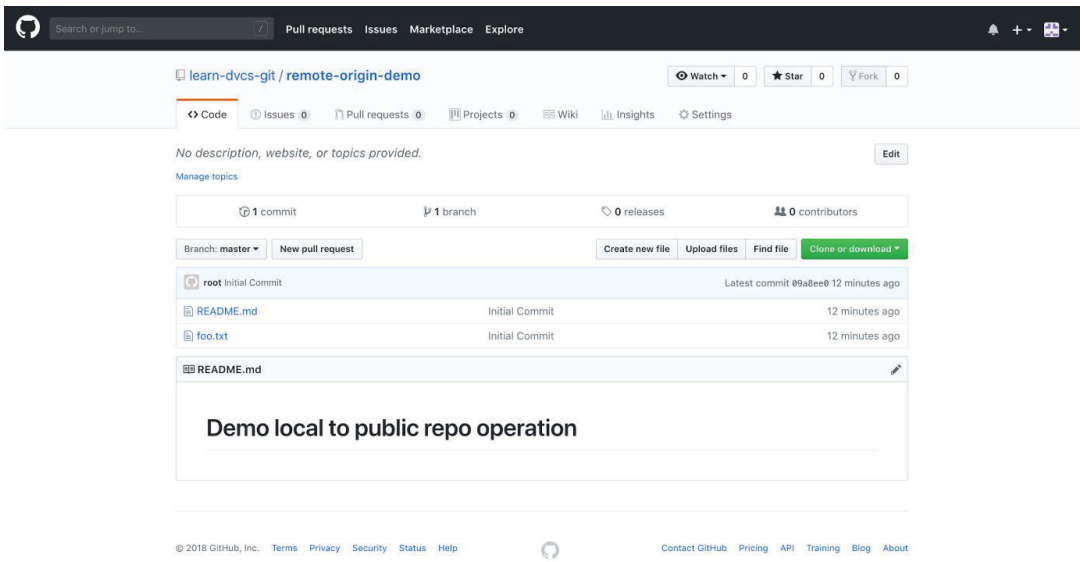
- Add local repository as remote for the repository at the specified URL

  $ git remote add origin https://github.com/learn-dvcs-git/remote-origin-demo.git
- Then push the changes in the local repository to the public repository

  $ git push -u origin master


```
$ git remote add origin https://github.com/learn-dvcs-git/remote-origin-demo.git
$ git push -u origin master
Username for 'https://github.com': learn-dvcs-git
Password for 'https://learn-dvcs-git@github.com':
Counting objects: 4, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 332 bytes | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:      https://github.com/learn-dvcs-git/remote-origin-demo/pull/new/master
remote:
To https://github.com/learn-dvcs-git/remote-origin-demo.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
$
```

- Now navigate to the URL specified in the previous command - https://github.com/learn-dvcs-git/remote-origin-demo.git



- Commits in the local repository will be reflected in the public repository

3. Working with Git

In this section, we will discuss the basic Git operations. To fully understand the Git operations, we need to be clear on the different file states and sections in Git and the basic Git workflow.

3.1. Git File States

→ Untracked

A new file in the working directory that Git isn't aware of, i.e., this file is not yet under version control.

→ Modified
This file has changed since the last commit and is yet to be committed to the Git database.

→ Staged
This file has been added to Git's version control or staging area to be part of next commit.

→ Committed
This file has been safely stored in the local database or the ***.git*** directory.

3.2. Git Project Section

→ Working Directory
Your new or modified files reside in this section. It is a single checkout of one version of your project.

→ Staging Area
This is where Git stores information about the changes that will be part of the next commit. This area is also known as ***index***.

→ Git Directory (Local Repository)
This is where the committed files reside, i.e., a snapshot of the files in the staging area is permanently stored in ***.git*** directory.

3.3. Basic Git Workflow

Action	Result
Add or Modify	Resides in working directory
Stage	Add snapshots of the staged files to the staging area
Commit	Move snapshots from staging area and stores permanently to the git directory

3.4. Basic Git Operations

3.4.1. git status

This command displays the state of the working directory and the staging area. It lets you see the changes to be committed, staged and the untracked files. It also instructs on the next operation to be executed.

$ git status

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   bar.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   foo.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        baz.txt

$
```

3.4.2. git add

This command tells Git to add the changes in the working directory to the staging area. The staged files will be considered for the next commit.

$ git add file-name

```
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        foo.txt

nothing added to commit but untracked files present (use "git add" to track)
$
$ git add foo.txt
$
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   foo.txt

$
```

3.4.3. git commit

This command tells Git to save the changes in the staging area to the local repository permanently.

$ git commit -m "Insert Commit Message Here"

```
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   foo.txt

$
$ git commit -m "Adding Foo"
[master (root-commit) a000825] Adding Foo
 1 file changed, 1 insertion(+)
 create mode 100644 foo.txt
$
$ git status
On branch master
nothing to commit, working directory clean
$
```

where -m option lets you specify the commit message inline. Use ***git help commit*** command to explore other available options.

3.4.4. git stage

This is a synonym for ***git add*** command that adds the changes in the working directory to the staging area.

$ git stage file-name

```
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        foo.txt

nothing added to commit but untracked files present (use "git add" to track)
$
$ git stage foo.txt
$
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   foo.txt

$
```


4. Git Configuration Files

Two of the most important Git configuration files are ***.gitattributes*** and ***.gitignore***.

4.1. .gitattributes

Git attributes are path-specific settings that applies to a subdirectory or subset of files. These settings can be defined in ***.gitattributes*** file in the root of your project directory or in ***.git/info/attributes*** file if you don't want it to be committed with your project.

Some of the common Git attributes are as follows:

4.1.1. Identifying Binary Files

To tell Git which files are binary, set below attributes in ***.gitattributes*** or ***.git/info/attributes***

E.g.:

**.png binary*
**.jpg binary*

Above setting implies that any file with ***.png*** and ***.jpg*** extension should be treated as binary files by Git.

4.1.2. Diffing Binary Files

Binary files cannot be directly compared. However, Git attributes can be used to tell Git how to convert your binary data to a text format to use normal diff command.

E.g.:

**.png diff=exif*

Above setting implies that any file with ***.png*** extension should use ***exif*** program to convert the binary data to text format to perform diff.

For above setting to work, download and install ***exiftool*** program and configure Git to use the tool.

E.g.:

$ git config diff.exif.textconv exiftool

4.1.3. Export Ignore

To tell Git not to include certain files and directories when generating an archive, you can set below git attribute.

E.g.:

test/ export-ignore

Above setting implies that the ***test*** subdirectory won't be included when ***git archive*** command is run. Note this subdirectory will still be committed to your project unless specified in ***.gitignore*** file.

4.1.4. Merge Strategy

You can tell Git to use different merge strategy for specific files in your project. For example, when there is a merge conflict you can tell Git to use your side of the merge over someone else's.

E.g.:

package-lock.json merge=ours

Above setting implies that when a merge conflict happens for ***package-lock.json*** file, original file is retained.

4.2. .gitignore

.gitignore file is used to tell Git to ignore specific untracked files, i.e., these files won't be staged or committed to the local repository.

Note: The files that are already tracked by Git won't be affected. This setting can be specified in ***.git/info/exclude*** or ***.gitignore*** in your project directory.

→ To stop ***.pyc*** files from being staged, add "*.pyc" to either of the afore-mentioned files.

$ echo ".pyc" >> .git/info/exclude*

Or

$ echo ".pyc" >> .gitignore*

```
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        config.py
        config.pyc

nothing added to commit but untracked files present (use "git add" to track)
$
$ echo "*.pyc" >> .git/info/exclude
$
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        config.py

nothing added to commit but untracked files present (use "git add" to track)
$
```

→ To stop tracking a file that is currently tracked, use ***git rm --cached***. This command deletes the file from the repository but remains in your working directory as ignored file.


```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   git-ignore-demo.egg

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        config.py

no changes added to commit (use "git add" and/or "git commit -a")
$
$ echo "*.egg" >> .git/info/exclude
$
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   git-ignore-demo.egg

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        config.py

no changes added to commit (use "git add" and/or "git commit -a")
$
$ git rm --cached git-ignore-demo.egg
rm 'git-ignore-demo.egg'
$
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    git-ignore-demo.egg

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        config.py

$
```

→ **Shared vs Private Git Ignore Rules**

Rule	File	Implication
Shared	.gitignore	This file is typically checked-in and is versioned like any other file in the repository.
Private	.git/info/exclude	This file is not versioned and distributed with the repository

5. Working with Git History

In this section, we will discuss ways of interacting with Git history.

5.1. git log

git log command is used to view the commit history. When no arguments are specified, it defaults to reverse chronological order, i.e., list the most recent commit first.

5.1.1. git log with no arguments

$ git log

```
$ git log
commit 599f4a909545ce6abe9393050eed1c8031c94f0a
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 05:20:47 2018 +0000

    Fourth Commit

commit 28c8fc10b9be6af5664374bac741952e0e40d3a8
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 05:20:19 2018 +0000

    Third Commit

commit 0b5a58cc07ab3c693221a4ad45cdd251677bded7
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 05:19:47 2018 +0000

    Second Commit

commit 7fbe9790658b3af3ae556b7b3982b2d4826f8144
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 05:19:19 2018 +0000

    First Commit
$
```

As you can see, this command lists each commit with its SHA-1 checksum, the author's name and email, the date written, and the commit message.

5.1.2. To view last *'x'* commits

$ git log -x

```
$ git log -2
commit 599f4a909545ce6abe9393050eed1c8031c94f0a
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 05:20:47 2018 +0000

    Fourth Commit

commit 28c8fc10b9be6af5664374bac741952e0e40d3a8
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 05:20:19 2018 +0000

    Third Commit
$
```

5.1.3. The -p or --patch option

To view the difference introduced in each commit, we can use the ***-p*** or ***--patch*** option

$ git log -p -1

```
$ git log -p -1
commit 17ccc963a30bc8c8f519650d58c4624d9305bf5f
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:23:27 2018 +0000

    Simplify the logic and invoke as standalone program

diff --git a/main.py b/main.py
index b0e30cb..b0d3888 100644
--- a/main.py
+++ b/main.py
@@ -11,4 +11,5 @@ def main():
        else:
                print "Not a Palindrome"

-main()
+if __name__ == '__main__':
+    main()
diff --git a/palindrome.py b/palindrome.py
index 12d10f9..3352f24 100644
--- a/palindrome.py
+++ b/palindrome.py
@@ -1,11 +1,7 @@
 #!/usr/bin/python

 def palindrome(word):
-       n = len(word)
-       for i in range(n):
-               if word[i] != word[n-1-i]:
-                       return False
+       if word != word[::-1]:
+               return False

        return True
-
-
$
```

5.1.4. The --oneline option

To view each commit on a single line, use ***--oneline*** option

$ git log --oneline

```
$ git log --oneline
17ccc96 Simplify the logic and invoke as standalone program
0ab13ce Add the import statement
1e467a5 Interact with user
6b0303b Program to check if a word is a palindrome
$
```

To browse through the list of available options for ***git log*** command, run ***$ git help log***. This opens up the man page for ***git log***.

5.2. Graphical History

Git comes with built-in GUI tools for committing (git-gui) and browsing (gitk).

5.2.1. git-gui

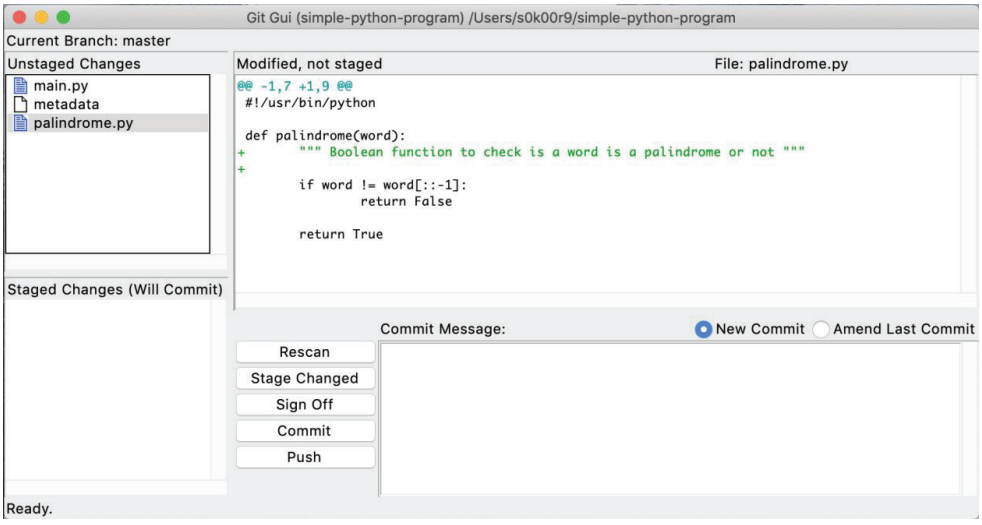
git gui focuses on allowing users to make changes to their repository by making new commits, amending existing ones, creating branches, performing local merges, and fetching/pushing to remote repositories. However, it does not show the project history.

Syntax:

$ git gui [<command>] [arguments]

where command and arguments are optional

$ git gui



To view the list of available options, run

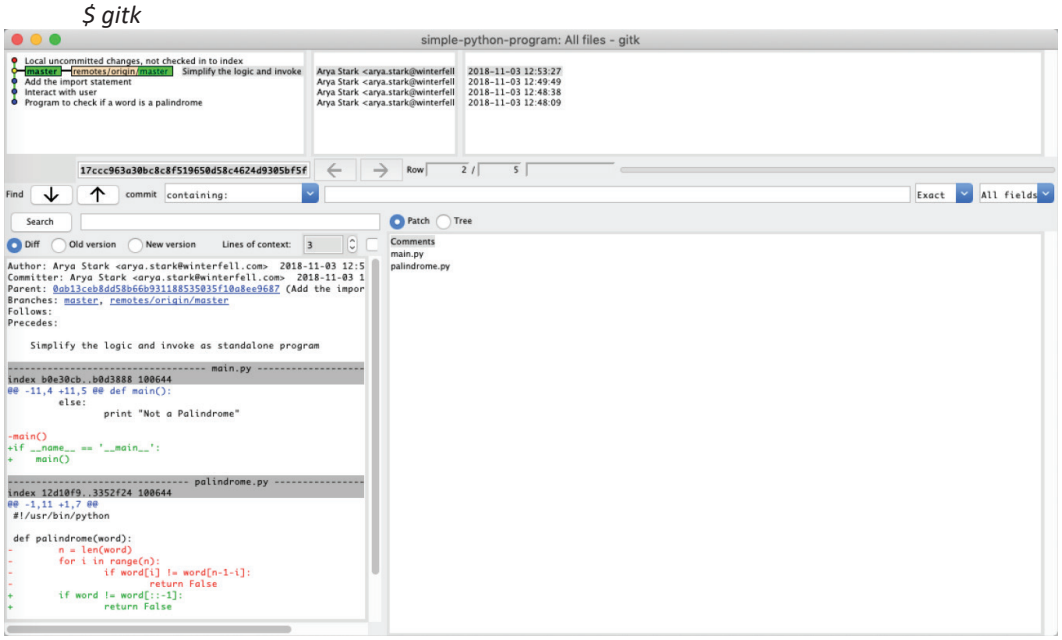
$ git help gui

This opens up the main page for *git gui*

5.2.2. gitk

gitk is the Git repository browser. Shows branches, commit history and file differences. This also includes visualizing the commit graph.

$ gitk

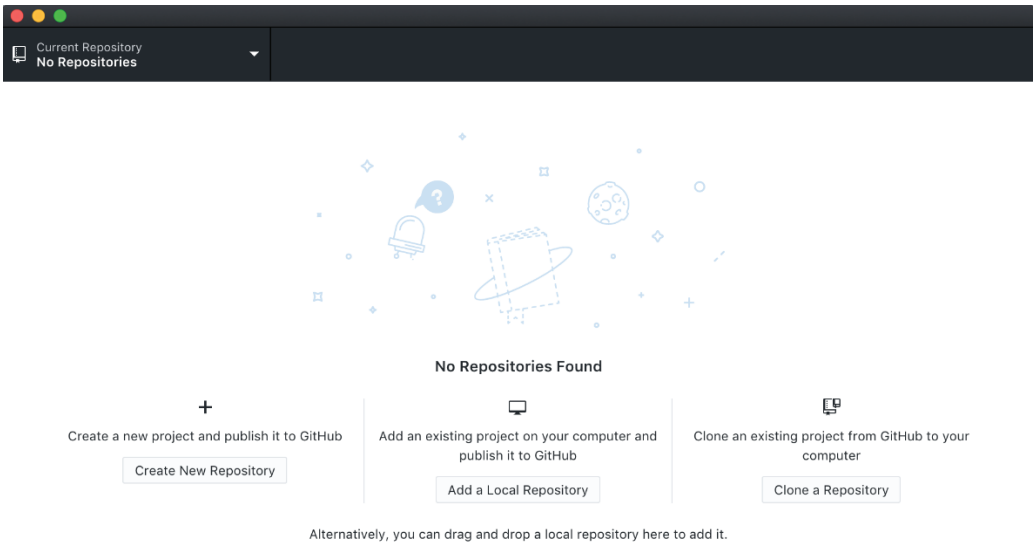


5.2.3. Third Party GUI Client

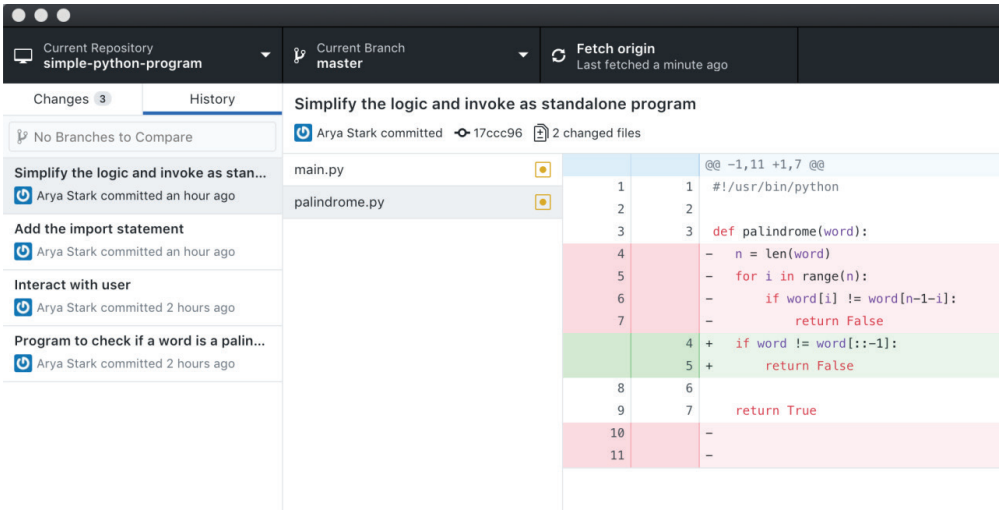
You can also view the commit history using a GUI client installed in the previous section. For this demo, we will use **GitHub Desktop** GUI client.

Steps:

- Open **GitHub Desktop** GUI client
- Optionally sign-in to your account and configure the username and email
- If this is your first-time using **GitHub Desktop** client, then you will see screen as below



- Click on **Add a Local Repository** or **Clone a Repository** and specify the path to your repository
- Now that your repo is added, on the left navigation bar, click on **History** tab
- You can browse through the list of commits on the left navigation bar and inspect the difference in each commit.



5.3. Undoing Changes in Git

In this section, we will discuss available ***undo*** Git strategies and commands.

5.3.1. Modify Last Commit

You can edit the last commit message or make additional changes to the commit by staging new changes and committing again with the ***--amend*** option.

E.g. 1: To amend commit message, use ***--amend*** option in the ***git commit*** command

$ git commit --amend

```
$ git log -1
commit ccddd9c649afe98f1bff59343955c36fe3586df6
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 10:17:37 2018 +0000

    Add new file
$
$ git commit --amend
[master 705f1df] Add project metadata like author info and version number
 Date: Sat Nov 3 10:17:37 2018 +0000
 1 file changed, 3 insertions(+)
 create mode 100644 metadata.txt
$
$ git log -1
commit 705f1df4e579ff468c9ee370aff4da4f5cfcaa70
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 10:17:37 2018 +0000

    Add project metadata like author info and version number
$
$
```

E.g. 2: Make additional changes to the last commit

$ git add new-updated-file

$ git commit --amend

```
$ git log -1
commit 705f1df4e579ff468c9ee370aff4da4f5cfcaa70
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 10:17:37 2018 +0000

    Add project metadata like author info and version number
$
$ git add main.py
$
$ git commit --amend
[master 7adc1cd] Add project metadata file and updates to main.py
 Date: Sat Nov 3 10:17:37 2018 +0000
 2 files changed, 5 insertions(+)
 create mode 100644 metadata.txt
$
$ git log -1
commit 7adc1cd7713697dae648101461f7488e344c7f99
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 10:17:37 2018 +0000

    Add project metadata file and updates to main.py
$
```

5.3.2. Unstaging a Staged File

Let us suppose you decide to unstage a staged file, you may do so by using the ***git reset*** command. The ***git status*** command will provide you with the help info to perform unstaging operation.

$ git reset HEAD file-to-be-unstaged

Note: With ***--hard*** option, you can also wipe out the uncommitted changes. Use this option with caution as you cannot recover the data.

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   foo.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        bar.txt

$
$ git reset HEAD foo.txt
$
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        bar.txt
        foo.txt

nothing added to commit but untracked files present (use "git add" to track)
$
```

5.3.3. Unmodifying a Modified File

Let us suppose you decide to revert the change to a file to its original state, you may do so by using the ***git checkout*** command. Again, the ***git status*** comes in handy to get the command info to perform the revert operation.

$ git checkout -- file-to-unmodify

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   palindrome.py

no changes added to commit (use "git add" and/or "git commit -a")
$
$ git diff
diff --git a/palindrome.py b/palindrome.py
index 3352f24..4ba57e7 100644
--- a/palindrome.py
+++ b/palindrome.py
@@ -1,6 +1,8 @@
 #!/usr/bin/python

 def palindrome(word):
+        """ Boolean function to check is a word is a palindrome or not """
+
         if word != word[::-1]:
                 return False

$
$ git checkout -- palindrome.py
$
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
$
```

You can see that the changes have been reverted. One important thing to note is that the changes made to the file are permanently gone and cannot be recovered. Use this command with caution.

5.3.4. Undo Commits with Git Checkout

Let us explore how to undo commits and continue to work on from an older snapshot without changing the current snapshot through the use of a new branch.

Steps:

- To begin, consider the below commit history for this illustration

```
$ git log
commit 7adc1cd7713697dae648101461f7488e344c7f99
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 10:17:37 2018 +0000

    Add project metadata file and updates to main.py

commit 17ccc963a30bc8c8f519650d58c4624d9305bf5f
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:23:27 2018 +0000

    Simplify the logic and invoke as standalone program

commit 0ab13ceb8dd58b66b931188535035f10a8ee9687
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:19:49 2018 +0000

    Add the import statement

commit 1e467a55c2132364026f96e64dbc617f9711bafa
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:18:38 2018 +0000

    Interact with user

commit 6b0303bfde5a942b8edbc1a45b8ee146d02db01e
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:18:09 2018 +0000

    Program to check if a word is a palindrome
$
```

- Let us suppose we want to undo all the commits from 0ab13ce commit id onwards. To do so, we use ***git checkout*** command on that commit id.

```
$ git checkout 0ab13ce
Note: checking out '0ab13ce'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 0ab13ce... Add the import statement
$
```

- As you can see in the above screenshot, we are now in ‘detached HEAD’ state, i.e., the HEAD points directly to this commit. To continue work from this commit onwards, perform ***git checkout -b <new-branch-name>***

```
$ git checkout -b new-branch-after-undo
Switched to a new branch 'new-branch-after-undo'
$
```

- Let us now verify if the undo operation is complete by checking the commit history

```
$ git log
commit 0ab13ceb8dd58b66b931188535035f10a8ee9687
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:19:49 2018 +0000

    Add the import statement

commit 1e467a55c2132364026f96e64dbc617f9711bafa
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:18:38 2018 +0000

    Interact with user

commit 6b0303bfde5a942b8edbc1a45b8ee146d02db01e
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:18:09 2018 +0000

    Program to check if a word is a palindrome
$
```

5.3.5. Undo Commits with Git Revert

The ***git revert*** command performs the undo operation by inversing the changes introduced in the last commit and appending a new commit with the resulting inverse content. This prevent Git from losing the commit history. This operation doesn't cause the 'detached HEAD' situation.

Steps:

- To begin, consider the below commit history for this illustration

```
$ git log --oneline
7adc1cd Add project metadata file and updates to main.py
17ccc96 Simplify the logic and invoke as standalone program
0ab13ce Add the import statement
1e467a5 Interact with user
6b0303b Program to check if a word is a palindrome
$
```

- Let us suppose we want to revert the last commit - 7adc1cd, run ***git revert HEAD***

```
$ git revert HEAD
[master f5b4e17] Revert "Add project metadata file and updates to main.py"
 2 files changed, 5 deletions(-)
 delete mode 100644 metadata.txt
$
$
```

- Reverting undoes a commit by adding new commit with inverse of changes. Let us now verify if the undo operation is complete by checking the commit history.

```
$ git log
commit f5b4e176fa7c9ace6a505cc604e09fa75fd2c70c
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 12:37:37 2018 +0000

    Revert "Add project metadata file and updates to main.py"

    This reverts commit 7adc1cd7713697dae648101461f7488e344c7f99.

commit 7adc1cd7713697dae648101461f7488e344c7f99
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 10:17:37 2018 +0000

    Add project metadata file and updates to main.py

commit 17ccc963a30bc8c8f519650d58c4624d9305bf5f
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:23:27 2018 +0000

    Simplify the logic and invoke as standalone program

commit 0ab13ceb8dd58b66b931188535035f10a8ee9687
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:19:49 2018 +0000

    Add the import statement

commit 1e467a55c2132364026f96e64dbc617f9711bafa
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:18:38 2018 +0000

    Interact with user

commit 6b0303bfde5a942b8edbc1a45b8ee146d02db01e
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:18:09 2018 +0000

    Program to check if a word is a palindrome
$
```

Note: Reverting doesn't overwrite the commit history.

To revert by *'n'* commits, run

$ git revert HEAD~n

To revert to a specific commit in the history, run

$ git revert commit-id

5.3.6. Undo Commits with Git Reset

The ***git reset*** command performs the undo operation by updating the ***HEAD*** and ***branch*** pointers to the specified commit leaving the rest of the commits from that commit onwards unreferenced. These unreferenced commits are orphan commits that will be permanently deleted during the internal garbage collection.

Note: This is the default behavior when no arguments are specified.
Steps:

- To begin, consider the below commit history for this illustration

```
$ git log
commit f5b4e176fa7c9ace6a505cc604e09fa75fd2c70c
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 12:37:37 2018 +0000

    Revert "Add project metadata file and updates to main.py"

    This reverts commit 7adc1cd7713697dae648101461f7488e344c7f99.

commit 7adc1cd7713697dae648101461f7488e344c7f99
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 10:17:37 2018 +0000

    Add project metadata file and updates to main.py

commit 17ccc963a30bc8c8f519650d58c4624d9305bf5f
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:23:27 2018 +0000

    Simplify the logic and invoke as standalone program

commit 0ab13ceb8dd58b66b931188535035f10a8ee9687
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:19:49 2018 +0000

    Add the import statement

commit 1e467a55c2132364026f96e64dbc617f9711bafa
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:18:38 2018 +0000

    Interact with user

commit 6b0303bfde5a942b8edbc1a45b8ee146d02db01e
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:18:09 2018 +0000

    Program to check if a word is a palindrome
$
```

- Let us suppose you want to undo the last commit - f5b4e17, run ***git reset*** on the previous commit to update the ***HEAD*** and ***branch*** pointers to it.

```
$ git reset 7adc1cd
Unstaged changes after reset:
M       main.py
D       metadata.txt
$
```

- As you can see in the above screenshot, resetting has undone the changes and have placed those changes in the working directory. Now you may discard these changes in the working directory using the ***git checkout -- file-name*** command or decide to build on those changes by updating them.

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   main.py
        deleted:    metadata.txt
```

- Let us now verify if the undo operation is complete by checking the commit history

```
$ git log
commit 7adc1cd7713697dae648101461f7488e344c7f99
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 10:17:37 2018 +0000

    Add project metadata file and updates to main.py

commit 17ccc963a30bc8c8f519650d58c4624d9305bf5f
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:23:27 2018 +0000

    Simplify the logic and invoke as standalone program

commit 0ab13ceb8dd58b66b931188535035f10a8ee9687
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:19:49 2018 +0000

    Add the import statement

commit 1e467a55c2132364026f96e64dbc617f9711bafa
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:18:38 2018 +0000

    Interact with user

commit 6b0303bfde5a942b8edbc1a45b8ee146d02db01e
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 07:18:09 2018 +0000

    Program to check if a word is a palindrome
$
```

Note: Resetting undoes by removing the commit references and removes any traces from the commit history.

5.3.7. Clean Untracked File

Git provides an option of cleaning the untracked files in the working directory.

$ git clean <option>

Options:

- -d ❼ Remove untracked directories in addition to untracked files
- -f ❼ Force clean operation
- -i ❼ Show what would be done and clean files interactively
- -n ❼ Don't actually remove anything, just show what would be done
- -x ❼ Don't use the standard ignore rules read from ***.gitignore*** (per directory) and $GIT_DIR/info/exclude
- -X ❼ Remove only files ignored by Git

Refer the below screenshot for the usage of the various option

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   tracked-file.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        untracked-dir/
        untracked-file.py
        untracked-ignored-file.py

no changes added to commit (use "git add" and/or "git commit -a")
$
$ echo "untracked-ignored-file.py" >> .git/info/exclude
$
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   tracked-file.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        untracked-dir/
        untracked-file.py

no changes added to commit (use "git add" and/or "git commit -a")
$
$ git clean -n
Would remove untracked-file.py
$
$ ls
tracked-file.py  untracked-dir  untracked-file.py  untracked-ignored-file.py
$
$ git clean -xdf
Removing untracked-dir/
Removing untracked-file.py
Removing untracked-ignored-file.py
$
$ ls
tracked-file.py
$
```

5.3.8. Remove File in Git Repository

The ***git rm*** command is used to remove files from the Git repository. This command operates on the current branch only. The removal event is only applied to the working directory and staging index trees. The file removal is not persisted to the repository history until a new commit is created.

The files being operated on must be identical to the files in the current HEAD. If there is a discrepancy between the HEAD version of a file and the staging index or working tree version, Git will block the removal. This block is a safety mechanism to prevent removal of in-progress changes.

6. Merge Resolution in Git

In this section, we will explore Git branching, merging, and resolving conflicts using the merge resolution workflow.

6.1. Git Branching

A branch is a representation of an independent line of development. You can think of a branch as a way of requesting a brand-new working directory, staging area, and the project history.

The ***git branch*** command lets you create, list, rename, and delete branches. It doesn't let you switch between branches or put a forked history back together again, i.e., merge back.

Let us see some Git branch operations in action.

6.1.1. List all Branches

$ git branch

Or

$ git branch --list

```
$ git branch
  develop
  feature
  hotfix
* master
$
```

The * before the branch name indicate that it is the current working branch.

6.1.2. Create a New Branch

$ git branch new-branch-name

```
$
$ git branch experimental
$
```


6.1.3. Delete a Branch

$ git branch -d branch-name

```
$ git branch -d experimental
Deleted branch experimental (was 0ab13ce).
$
```

This is a “safe” operation in that Git prevents you from deleting the branch if it has unmerged changes.

6.1.4. Rename Current Branch

$ git branch -m new-branch-name

```
$ git branch
  develop
* feature
  hotfix
  master
$
$ git branch -m experimental
$
$ git branch
  develop
* experimental
  hotfix
  master
$
```

6.1.5. List all Remote Branches

$ git branch -a

```
$ git branch -a
  develop
* experimental
  hotfix
  master
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
$
```

6.1.6. Switch to Different Branch

$ git checkout branch-name

```
$ git checkout master
Switched to branch 'master'
$
```

6.2. Git Merging

Merging refers to integrating the independent lines of development into a single branch.

The ***git merge*** command is used to combine two branches. The merge commits are unique because they have two parent commits.

Let us see some Git merge operations in action.

6.2.1. Fast Forward Merge

A fast-forward merge can occur when there is a linear path from the current branch tip to the target branch, i.e., the branches have not diverged. Instead of "actually" merging the branches, Git moves the current branch tip to the target branch tip. In this case, no merge commit will be recorded in the commit history unless ***--no-ff*** option is provided.

E.g. 1: Fast Forward Merge without Merge Commit

Steps:

- To merge ***feature*** branch into the ***master*** branch, let us consider below commit history for illustration

→ Master Branch Commit History

```
$ git log --pretty=short
commit 0ab13ceb8dd58b66b931188535035f10a8ee9687
Author: Arya Stark <arya.stark@winterfell.com>

    Add the import statement

commit 1e467a55c2132364026f96e64dbc617f9711bafa
Author: Arya Stark <arya.stark@winterfell.com>

    Interact with user

commit 6b0303bfde5a942b8edbc1a45b8ee146d02db01e
Author: Arya Stark <arya.stark@winterfell.com>

    Program to check if a word is a palindrome
$
```

→ Feature Branch Commit History

```
$ git log --pretty=short
commit 1278cfab0fdaf2a73ba6aecb441f29d52b99bf23
Author: Arya Stark <arya.stark@winterfell.com>

    New feature - reverse as a function

commit 0ab13ceb8dd58b66b931188535035f10a8ee9687
Author: Arya Stark <arya.stark@winterfell.com>

    Add the import statement

commit 1e467a55c2132364026f96e64dbc617f9711bafa
Author: Arya Stark <arya.stark@winterfell.com>

    Interact with user

commit 6b0303bfde5a942b8edbc1a45b8ee146d02db01e
Author: Arya Stark <arya.stark@winterfell.com>

    Program to check if a word is a palindrome
$
```

- Next, switch to the ***master*** branch and run ***git merge*** command


```
$ git checkout master
Switched to branch 'master'
$
$ git merge feature
Updating 0ab13ce..1278cfa
Fast-forward
 reverse.py | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 reverse.py
$
```

- Let us now verify the merge operation by viewing the commit history

```
$ git log --pretty=short
commit 1278cfab0fdaf2a73ba6aecb441f29d52b99bf23
Author: Arya Stark <arya.stark@winterfell.com>

    New feature - reverse as a function

commit 0ab13ceb8dd58b66b931188535035f10a8ee9687
Author: Arya Stark <arya.stark@winterfell.com>

    Add the import statement

commit 1e467a55c2132364026f96e64dbc617f9711bafa
Author: Arya Stark <arya.stark@winterfell.com>

    Interact with user

commit 6b0303bfde5a942b8edbc1a45b8ee146d02db01e
Author: Arya Stark <arya.stark@winterfell.com>

    Program to check if a word is a palindrome
$
```

Notice that the commit history of ***master*** branch is exactly the same as that of the ***feature*** branch. It looks as though there was no merge operation performed.

E.g. 2: Fast Forward Merge with Merge Commit

Steps:

- To merge ***feature*** branch into the ***master*** branch, let us consider below commit history for illustration

  → Master Branch Commit History

```
$ git log --pretty=short
commit 0ab13ceb8dd58b66b931188535035f10a8ee9687
Author: Arya Stark <arya.stark@winterfell.com>

    Add the import statement

commit 1e467a55c2132364026f96e64dbc617f9711bafa
Author: Arya Stark <arya.stark@winterfell.com>

    Interact with user

commit 6b0303bfde5a942b8edbc1a45b8ee146d02db01e
Author: Arya Stark <arya.stark@winterfell.com>

    Program to check if a word is a palindrome
$
```

→ Feature Branch Commit History

```
$ git log --pretty=short
commit 1278cfab0fdaf2a73ba6aecb441f29d52b99bf23
Author: Arya Stark <arya.stark@winterfell.com>

    New feature - reverse as a function

commit 0ab13ceb8dd58b66b931188535035f10a8ee9687
Author: Arya Stark <arya.stark@winterfell.com>

    Add the import statement

commit 1e467a55c2132364026f96e64dbc617f9711bafa
Author: Arya Stark <arya.stark@winterfell.com>

    Interact with user

commit 6b0303bfde5a942b8edbc1a45b8ee146d02db01e
Author: Arya Stark <arya.stark@winterfell.com>

    Program to check if a word is a palindrome
$
```

- Next, switch to the ***master*** branch and run ***git merge*** command with ***--no-ff*** option

```
$ git checkout master
Switched to branch 'master'
$
$ git merge --no-ff feature
Merge made by the 'recursive' strategy.
 reverse.py | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 reverse.py
$
```

- Let us now verify the merge operation by viewing the commit history


```
$ git log --pretty=short
commit 4aa3ee37d05fb14440773821454fb94fada1e72f
Merge: 0ab13ce 1278cfa
Author: Arya Stark <arya.stark@winterfell.com>

    Merge branch 'feature'

commit 1278cfab0fdaf2a73ba6aecb441f29d52b99bf23
Author: Arya Stark <arya.stark@winterfell.com>

    New feature - reverse as a function

commit 0ab13ceb8dd58b66b931188535035f10a8ee9687
Author: Arya Stark <arya.stark@winterfell.com>

    Add the import statement

commit 1e467a55c2132364026f96e64dbc617f9711bafa
Author: Arya Stark <arya.stark@winterfell.com>

    Interact with user

commit 6b0303bfde5a942b8edbc1a45b8ee146d02db01e
Author: Arya Stark <arya.stark@winterfell.com>

    Program to check if a word is a palindrome
$
```

The important thing to notice here is the explicit merge commit in addition to the commits in the target branch.

6.2.2. 3-way Merge

The 3-way merge occur when the branches have diverged. In this case, Git uses 3 commits to generate the merge commit: the two branch tips and their common ancestor.

Steps:

- To merge ***feature*** branch into the ***develop*** branch, let us consider below commit history for illustration

  → Develop Branch Commit History

```
$ git log --pretty=short
commit ec68f48f52bdeb80c4ab12b10c7adfcebd0b2bab
Author: Arya Stark <arya.stark@winterfell.com>

    Add more comments to increase readability

commit 17ccc963a30bc8c8f519650d58c4624d9305bf5f
Author: Arya Stark <arya.stark@winterfell.com>

    Simplify the logic and invoke as standalone program

commit 0ab13ceb8dd58b66b931188535035f10a8ee9687
Author: Arya Stark <arya.stark@winterfell.com>

    Add the import statement

commit 1e467a55c2132364026f96e64dbc617f9711bafa
Author: Arya Stark <arya.stark@winterfell.com>

    Interact with user

commit 6b0303bfde5a942b8edbc1a45b8ee146d02db01e
Author: Arya Stark <arya.stark@winterfell.com>

    Program to check if a word is a palindrome
$
```

→ Feature Branch Commit History

```
$ git log --pretty=short
commit 1278cfab0fdaf2a73ba6aecb441f29d52b99bf23
Author: Arya Stark <arya.stark@winterfell.com>

    New feature - reverse as a function

commit 0ab13ceb8dd58b66b931188535035f10a8ee9687
Author: Arya Stark <arya.stark@winterfell.com>

    Add the import statement

commit 1e467a55c2132364026f96e64dbc617f9711bafa
Author: Arya Stark <arya.stark@winterfell.com>

    Interact with user

commit 6b0303bfde5a942b8edbc1a45b8ee146d02db01e
Author: Arya Stark <arya.stark@winterfell.com>

    Program to check if a word is a palindrome
$
```

- Next, switch to the ***develop*** branch and run ***git merge*** command

```
$ git checkout develop
Switched to branch 'develop'
$
$ git merge feature
Merge made by the 'recursive' strategy.
 reverse.py | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 reverse.py
$
```


- Let us now verify the merge operation by viewing the commit history

```
$ git log --pretty=short
commit 802b6b5482cc8212792b9a77dbe0bb52e95384fe
Merge: ec68f48 1278cfa
Author: Arya Stark <arya.stark@winterfell.com>

    Merge branch 'feature' into develop

commit 1278cfab0fdaf2a73ba6aecb441f29d52b99bf23
Author: Arya Stark <arya.stark@winterfell.com>

    New feature - reverse as a function

commit ec68f48f52bdeb80c4ab12b10c7adfcebd0b2bab
Author: Arya Stark <arya.stark@winterfell.com>

    Add more comments to increase readability

commit 17ccc963a30bc8c8f519650d58c4624d9305bf5f
Author: Arya Stark <arya.stark@winterfell.com>

    Simplify the logic and invoke as standalone program

commit 0ab13ceb8dd58b66b931188535035f10a8ee9687
Author: Arya Stark <arya.stark@winterfell.com>

    Add the import statement

commit 1e467a55c2132364026f96e64dbc617f9711bafa
Author: Arya Stark <arya.stark@winterfell.com>

    Interact with user

commit 6b0303bfde5a942b8edbc1a45b8ee146d02db01e
Author: Arya Stark <arya.stark@winterfell.com>

    Program to check if a word is a palindrome
```

Notice that we have two additional commits: one from ***feature*** branch and another is the merge commit itself.

6.3. Resolving Conflict with Merge Resolution Workflow

A conflict arises when two separate branches have made edits to the same line in a file, or when a file has been deleted in one branch but edited in the other. In this situation, Git won't be able to figure out which version to use and stops right before merge commit so that you can resolve the conflicts manually.

Note: The merge conflicts will only occur in the event of a 3-way merge. It’s not possible to have conflicting changes in a fast-forward merge.

Let us now see how to resolve merge conflicts manually.

Steps:

- Let us suppose both the ***master*** and ***hotfix*** branches have changes to the same lines in a file *say palindrome.py*

→ Master Branch

```
$ git checkout master
Switched to branch 'master'
$
$ git log
commit 36743cd73b6803ecf0321f2f0ac515afd21fdea7
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 18:08:15 2018 +0000

    Iterative logic

commit 8f305c4b52c7e79e4b28eddf686b3cbd9cc246b8
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 18:00:50 2018 +0000

    Program to check if a word is a palindrome
$
$ cat palindrome.py
#!/usr/bin/python

def palindrome(word):
        n = len(word)
        for i in range(n):
                if word[i] != word[n-1-i]:
                        return False

        return True
$
```

→ Hotfix Branch

```
$ git checkout hotfix
Switched to branch 'hotfix'
$
$ git log
commit 7d05e68e9a36f1a81713c28ae9f3fcc6897504aa
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 18:02:08 2018 +0000

    Reverse and Compare logic

commit 8f305c4b52c7e79e4b28eddf686b3cbd9cc246b8
Author: Arya Stark <arya.stark@winterfell.com>
Date:   Sat Nov 3 18:00:50 2018 +0000

    Program to check if a word is a palindrome
$
$ cat palindrome.py
#!/usr/bin/python

def palindrome(word):
        """ Boolean function to check is a word is a palindrome or not """

        if word != word[::-1]:
                return False

        return True
$
```

- Let us now try to merge the ***hotfix*** branch into the ***master*** branch

```
$ git checkout master
Switched to branch 'master'
$
$ git merge hotfix
Auto-merging palindrome.py
CONFLICT (content): Merge conflict in palindrome.py
Automatic merge failed; fix conflicts and then commit the result.
$
```

- As you can see in the above screenshot, we have a merge conflict that needs to be manually resolved. Let us suppose we decide to keep changes in the hotfix branch, we can do so by following below steps.
    - Identify conflicting files

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   palindrome.py

no changes added to commit (use "git add" and/or "git commit -a")
$
```

    - Open palindrome.py in your favorite editor and remove unwanted changes

→ Before Fix

```
$ cat palindrome.py
#!/usr/bin/python

def palindrome(word):
<<<<<<< HEAD
        n = len(word)
        for i in range(n):
                if word[i] != word[n-1-i]:
                        return False

        return True
=======
        """ Boolean function to check is a word is a palindrome or not """

        if word != word[::-1]:
                return False

        return True
>>>>>>> hotfix
$
```

→ After Fix

```
$ cat palindrome.py
#!/usr/bin/python

def palindrome(word):
        """ Boolean function to check is a word is a palindrome or not """

        if word != word[::-1]:
                return False

        return True
$
```

- Now that we have manually fixed the merge conflict, we need to stage the conflicted files to tell Git that there are resolved and perform a commit operation to generate the merge commit.

```
$ git branch
  hotfix
* master
$
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   palindrome.py

no changes added to commit (use "git add" and/or "git commit -a")
$
$ git add palindrome.py
$ git commit -m "Manually fixed the merge conflict"
[master 125eb6a] Manually fixed the merge conflict
$
$ git status
On branch master
nothing to commit, working directory clean
$
$ git log --pretty=short
commit 125eb6a8c2dd5f3f6dca25d848d5fc7db7083359
Merge: 36743cd 7d05e68
Author: Arya Stark <arya.stark@winterfell.com>

    Manually fixed the merge conflict

commit 36743cd73b6803ecf0321f2f0ac515afd21fdea7
Author: Arya Stark <arya.stark@winterfell.com>

    Iterative logic

commit 7d05e68e9a36f1a81713c28ae9f3fcc6897504aa
Author: Arya Stark <arya.stark@winterfell.com>

    Reverse and Compare logic

commit 8f305c4b52c7e79e4b28eddf686b3cbd9cc246b8
Author: Arya Stark <arya.stark@winterfell.com>

    Program to check if a word is a palindrome
$
```

Notice that the merge conflict resolution is similar to the normal Git workflow, i.e., modify/stage/commit.

Release Notes

B. TECH CSE with Specialization in DevOps

Semester Two -Year 01

Release Components.
Facilitator Guide, Facilitator Course Presentations, Student Guide and Mock exams.

Current Release Version.
1.0.0

Current Release Date.
6 Feb 2019

Course Description.
Xebia, has been recognized as a leader in DevOps by Gartner and Forrester and this course is created by Xebia to equip students with set of practices, methodologies and tools that emphasizes the collaboration and communication of both software developers and other information-technology (IT) professionals while automating the process of software delivery and infrastructure changes.

The language used in this course is US English. Our sources of reference for grammar, syntax, and mechanics are from The Chicago Manual of Style, The American Heritage Dictionary, and the Microsoft Manual of Style for Technical Publications.

Bugs reported	Not applicable for version 1.0.0
Next planned release	Version 2.0.0 Sept. 2019