Git Tutorial

Prepared by Mr. Ahmad Osman

OVERVIEW & PURPOSE

In this Tutorial, the participant will learn Git from scratch.

EDUCATION STANDARDS

1. The participant should know how to install Git in many Operating Systems.
2. The Participant should know the basics about Linux Terminal like creating folders, files, and editors

OBJECTIVES

1. Learn how to configure git repository.
2. Learn how to inspect the Git repository using status and log checks.
3. Learn how to communicate with GitHub from the local repository.
4. Learn how to modify, add and commit artifacts
5. Learn Git branching basics
6. Learn GitHub and how to use and manage the web-based Git repository
7. Learn HTTPS and SSH connection with GitHub
8. Learn basic file management in the git bash environment
9. Learn how to undo changes in Git repository through checkout, reset, revert and clean

VERIFICATION

Steps to check for student understanding

1. Create a repo.
2. Corporate with other participants using Git.
3. Solve the merging problems

Configuring author and email

View Git configurations

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

Configuration username

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

Configuration user Email

```
git config --global user.email "User Email"
```

Example

```
# home directory
$ pwd
$ mkdir git-fast
$ cd git-fast
$ pwd
$ git config --global --list
$ git config --global user.name "ahmad-dci"
$ git config --global --list
$ git config --global user.email "ahmad.odman@digitalcareerinstitute.org"
$ git config --global --list

# vi is the editor, ~ is the home directory
$ vi ~/.gitconfig
$ cat .gitconfig
```

Initializing an empty repository

The cursor should refer to the folder that contains the Repo

```
git init
```

Example

```
$ pwd
$ mkdir myRepoFromScratch
$ cd myRepoFromScratch
$ pwd

# notice the creation of the .git directory
```

```
$ git init

# "ls -al" command lists all objects, including hidden
$ ls -al

# without using any editor
$ echo "this is my first file in empty repository" >>
firstFileUsingEcho.txt
$ ls
$ cat firstFileUsingEcho.txt

 # create the file and enter some contents
$ vi  secondFileUsingVI.txt
$ ls
$ cat secondFileUsingVI.txt
```

Existing unversioned project to the repository

Download an initialised project from http://www.initializr.com

```
$ pwd
 # copy the downloaded file
$ cp /f/ahmad/Downloads/initializr-verekia-4.0.zip
# unzipping will create a folder called "initilizr"
$ unzip initializr-verekia-4.0.zip
$ ls -al
# remove the copied file
$ rm initializer-verekia-4.0.zip
# rename directory
$ mv initializr/ myRepoFromExistingSource
# now we need to add source control to the unversioned repo
"myRepoFromExistingSource"
$ cd myRepoFromExistingSource
# Initialized existing source
$ git init
# .git folder can be seen with ls -al
$ ls -al
```

Command Summary - configuration & repository creation

```
# Typically, you'll want to use the --global flag to set configuration
options for the current user.
# Define the author name to be used for all commits by the current user.
$ git config --global user.name <name>
```

```
# Define the author email to be used for all commits by the current user.
$ git config --global user.email <email>
# For listing all the global configuration at the user level
$ git config --global --list
# Transforms the current directory into a Git repository; the command is
same for creating a repo from .. contd...
# scratch or convert an existing unversioned code base into a git
repository
$ git init
```

Command Summary - Accessing Git Help system

```
# general help
$ git help

# lists sub-commands
$ git help -a

# lists concept guides
$ git help -g

# read about a specific sub-command
$ git help <command>

# read about a specific concept
$ git help <concept>
```

Copying a GitHub repository

```
$ cd <project_directory>
$ git clone https://github.com/<user>/<example_project>
$ cd example_project
$ ls -al
```

Committing changes in Git

Adding your changes

```
# home directory
$ pwd
$ cd git-fast
$ ls
$ cd myRepoFromScratch
$ ls

# create a file and add some content
$ vi demoFile1
$ cat demoFile1
$ git status
$ git add demoFile1
$ git rm --cached demoFile1
$ git status
```

Committing changes

```
# home directory
$ pwd
$ git status
$ git add demofile1
$ git status
$ git commit -m "our first commit in this course"
$ git status
```

Command Summary - add and commit

```
$ git add
$ git commit -m "message goes here"
```

How to check your repo status

```
# cd to /c/Users/Lenovo-PC/git-fast/myRepoFromScratch
$ pwd
$ ls

# start with a clean working directory
$ git status

# create a file and add some contents such as "I want to shed 15 kg 5
weeks"
$ vi weightLossChart

# create a file and add some contents such as "I want to add more green
vegetables to my diet"
$ vi dietChart
$ git status

 # same as "git status" since "long" option is the default one (compare
output)
$ git status --long
$ git status -s                                 # status "??" for untracked
file
$ git add weightLossChart
$ git status -s                                 # status "A"
$ git commit -m "1st commit for weightLossChart"
$ git status -s
$ vi weightLossChart                  # make some changes to the file
$ git status -s                               # status "M"
$ git add weightLossChart
$ git status -s                                 # status "M"
$ git commit -m "2nd commit for weightLossChart"
$ git status -s
$ mv weightLossChart weightLossChart2
$ git status -s                              # status "D"
```

How to check commit history

```
# displays the entire commit history using the default formatting
$ git log
# oneline condensed view of each commit history
$ git log --oneline
# Only display commits that include the specified file
$ git log <file>
# Show only commits that occur between <since> and <until>. Both arguments
can be either a commit ID ...contd.
# a branch, name, HEAD, or any other kind of revision reference.
$ git log <since>..<until>
# Limit the number of commits by <limit>. For example, git log -n 3 will
display only 3 commits.
$ git log -n <limit>
```

Command Summary - status & log

```
# provides a status report of the repo
$ git status
# displays the entire commit history using the default formatting
$ git log
 # oneline condensed view of each commit history
$ git log --oneline
# Only display commits that include the specified file
$ git log <file>
# Show only commits that occur between <since> and <until>. Both arguments
```

```
can be either a commit ID, a branch
# name, HEAD, or any other kind of revision reference.
$ git log <since>..<until>
# Limit the number of commits by <limit>. For example, git log -n 3 will
display only 3 commits.
$ git log -n <limit>
```

Command Summary (Undoing changes in a Git repository)

Checking out commits in a Git repository Part 1

```
$ cd git-fast
# created a fresh git repository named "demo-checkout-commit" using
initializr.zip
# Build up a commit history by modifying the file robots.txt, say 5 times
and committing each time
# Let us assume we have one commit whose id/hash is 91770af
# the choice of commit id 91770af is arbitrary

# note where the HEAD is now (since HEAD will change later while doing a
checkout)
$ git log --oneline

# clear the screen
# The command below will put you in a detached HEAD state.
# HEAD no longer points to a branch; it points directly to a commit.
$ git checkout 91770af
```

Checking out commits in a Git repository Part 2

```
# you are in detached HEAD state and inside git repository
(demo-checkout-commit)
# You can see only commits till commit-id 91770af
$ git log --oneline

# now edit and do an express commit for file robots.txt
# note down your new commit id/hash (Say ID1)
$ git log --oneline

$ git checkout master

# changes made in the detached HEAD state is no more there
# the new commit ID1 is not visible
$ git log --oneline

# again you are in the detached HEAD state
```

```
$ git checkout 91770af

# the new commit ID1 is not visible which means earlier changes...
# in the detached HEAD state were not preserved
$ git log --oneline

# now we want to retain our commit i.e, preserve our changes
# make changes in the robots.txt and do an express commit

# create a branch named weird-experiment
$ git branch  weird-experiment

$ git checkout master
# the new commit id is not visible here
# you can check the file robots.txt using cat command also
$ git log --oneline

# new commit id is visible now
# you can examine the file robots.txt using the cat command
$ git checkout weird-experiment
```

Checking out files

```
# cd to the git repository (demo-checkout-commit) used in the previous
lecture
$ git log --online

# Let us assume we have the commit-id 91770af

# step-2: examine the current contents of the file
$ cat robots.txt

# now we will able to see the content of the file as at commit-id 91770af
$ git checkout 91770af robots.txt

# this will show the file has been added to the staging area
$ git status

# step-5: examine the contents of the file (should be different from
step-2)
$ cat robots.txt

# if we do not want to revert back the file robots.txt
$ git checkout HEAD robots.txt
```

```
# you will see a clean working directory
$ git status

# now we want to revert back to the previous version of robots.txt as at
commit-id 91770af
$ git checkout 91770af robots.txt
$ git commit -am "reverting backfile state to commit 91770af"

# clean working directory
$ git status


# file is reverted and the contents are as seen in step-5
$ cat robots.txt
```

Reverting changes

```
# cd into repository "undo-demo-git-repo"

# add the text say, "line 5"
$ vim checkoutfile.txt

# modified stage visible
$ git status

# commit the above change
$ git commit -am checkoutfile.txt "commit message..."

$ git log --oneline
$ cat checkoutfile.txt

# Now let's assume the earlier commit is a buggy one and so we will revert
it
# revert command introduces a new commit in a safe manner (opens the
default editor)
$ git revert HEAD

# check commit history - a new commit for revert
$ git log --oneline

# note the change made earlier has been reverted "line 5" is not visible
$ cat checkoutfile.txt
```

Resetting Git repository Part1

```
# we are in repository git-reset-demo-1 with 2 files goldies-time-table and
jimmys-time-table
# the repository has an existing commit history

# now edit file jimmys-time-table and add a line at the end
$ vim jimmys-time-table

# add the change to the staging area
$ git add jimmys-time-table

# change in the staging area is visible
$ git status

# undoes changes in the staging area
$ git reset jimmys-time-table

# changes in the staging area are removed; file in modified state
$ git status

# the changed in the working directory is intact
$ cat jimmys-time-table
```

Resetting Git repository Part2

```
# we are in repository git-reset-demo-2 (this repo is clone
of git-reset-demo-1)

# edit the file and add one line at the end
$ vim jimmys-time-table

# edit the file and add one line at the end
$ vim goldies-time-table

# files are in the modified state
$ git status

# add both files to the staging area
$ git add .

# changes visible in the staging area
```

```
$ git status

# "reset" command removes changes in staging area leaving
the working directory intact
$ git reset

# both files in the modified state
$ git status

#######################################
# git reset demo using the --hard option
# we are still in git-reset-demo-2 repository(this repo is
clone of git-reset-demo-1)

# files are in the modified state
$ git status

# add both files to the staging area
$ git add .

# changes visible in the staging area
$ git status

# --hard option removes all changes in the working
directory and staging area
$ git reset --hard

# the working directory is clean since all changes have
been reset in working dir and staging
$ git status
```

Resetting Git repository Part3

```
# we are inside git-reset-demo-4 repository (this repo is clone of
git-reset-demo-1)

# Let's say, we have 5 commits in the commit history
# let' say, the 5th commit-id (HEAD) is 9166e4f and 4th commit-id is
d21a539
$ git log --oneline

# let's reset the repository to 4th commit-id
$ git reset d21a539

# the 5th commit-id(9166e4f) has been removed from history and the HEAD is
now at d21a539
$ git log --oneline

# the files are in modified state and changes in staging area has been
removed
# the files are jimmys-time-table and goldies-time-table
$ git status

# now we will commit the above changes in working dir in 2 smaller chunks
$ git add jimmys-time-table
$ git commit -m "commit message....."

# now only goldies-time-table changes are visible in modified state
$ git status

$ git add goldies-time-table
$ git commit -m "commit message....."

# clean working directory
$ git status

$ git log --oneline
```

```
# we are in git-reset-demo-5 repository (this repo is clone of
git-reset-demo-1)

# let's say there are 5 commits
$ git log --oneline

# now we will reset to the 3rd commit-id a53f51c (2 commits before HEAD)
with the --hard option
# this command will remove 4th & 5th commit-ids as well removed all changes
in working dir and staging area
$ git reset --hard a53f51c

# the 4th and 5th commit-ids has been removed from history
$ git log --oneline

# examine the file contents to confirm things
# the lines pertaining to 4th and 5th commit-ids has been removed
$ cat goldies-time-table
$ cat jimmys-time-table
```

Cleaning Git repository

```
$ cd git-fast
$ mkdir git-clean-demo
$ cd git-clean-demo

# create an empty repository
$ git init

# create few empty files
$ touch clean-demo-file
$ touch clean-demo-tracked-file

$ git add clean-demo-tracked-file
$ git commit -m "commit tracked file"

# shows clean-demo-file as an untracked file
$ git status

# -n option allows to make a dry run of "git clean" command
$ git clean -n

#################################################
```

```
# -f option (-f means force) removes untracked files from the current
directory
# so this will remove clean-demo-file
$ git clean -f

#################################################

$ mkdir clean-demo-dir
$ cd clean-demo-dir

# create an untracked file
$ touch clean-demofile-2

$ cd ..

# removes the untracked file in path clean-demo-dir
$ git clean -f clean-demo-dir/

#confirm untracked file cleaning
$ cd clean-demo-dir

# "ls" produces no results meaning the untracked file was cleaned
$ ls

#################################################
$ cd ..
$ ls
$ touch clean-demo-file-2
$ ls

# -df option cleans both untracked files and directories
$ git clean -df
$ ls

#################################################
# create .gitignore file and add the file name "clean-demo-file-5"
$ vim .gitignore
$ git add .gitignore
$ git commit -m "committing .gitignore file"

$ touch clean-demo-file-4
$ touch clean-demo-file-5
$ ls -al

# -xf option cleans all untracked files including those mentioned in
```

```
.gitignore file
$ git clean -xf

$ ls

```

pull and push to GitHub

```
$ git pull

# cloning generally sets up both the names "origin and master" for you
automatically
$ git push origin master
```