Central vs Distributed (Central+ copy on each user PC) VersionControlSystems

1. Check Git version

*git --version*

* 1. Setting initial user settings, recognition, identification

*git config –-global user.name “name”*

*git config –-global user.email “email address”*

* 1. Listing all configuration values

*git config –list*

* 1. Getting help/manuals

*git help <verb>*

*git <verb> --help*

1. turn directory into a Git project by/ initialise a repository from existing code (need to be inside the directory containing the file):

*git init*

sets up all the tools Git needs to begin tracking changes

Initalized an empty git repository in /home/ccuser/workspace/sorcerers code/.git/

creates the .git file in the directory



committing changes to save them

1.1 to reverse it/stop tracking project

*rm –rf .git*

1. check the status of changes, always run after git init

*git status*

1. Git to start tracking **a** file need to be added to the staging area.

*git add filename*

* can add multiples, just list consecutively

*git add filename-1.txt filename-2.txt*

1. If file is tracked, check the differences between the working directory and the staging area

*git diff filename*

1. commit permanently stores changes from the staging area inside the repository

*git commit –m*

*git commit -m "Complete first line of dialogue"*

Standard Conventions for Commit Messages:

* Must be in quotation marks
* Written in the present tense
* Should be brief (50 characters or less) when using –m

1. Commits are stored chronologically in the repository and can be viewed with:

*git log*

A 40-character code, called a *SHA*, which uniquely identifies the commit.

1. commit you are currently on is known as the HEAD commit.

*git show HEAD filename*

display everything the *git log* displays for the HEAD commit, plus all the file changes that were committed

1. restore the file in your working directory to look exactly as it did when you last made a commit.

*git checkout HEAD filename*

1. *unstage* that file from the staging area

*git reset HEAD filename*

command *resets* the file in the staging area to be the same as the HEAD commit. It does not discard file changes from the working directory, it just removes them from the staging area.

1. Command works by using the first 7 characters of the SHA of a previous commit. For example, if the SHA of the previous commit is 5d692065cf51a2f50ea8e7b19b5a7ae512f633ba; HEAD is now set to that previous commit.

*git reset commit\_SHA*

*git reset 5d69206*

1. \* (asterisk) showing you what branch you’re on.

*git branch*

1. create a new branch

*git branch new\_branch*

1. switch to the new branch with

*git checkout branch\_name*

1. *merging* the branch into master with:

*git merge branch\_name*

1. delete the specified branch from your Git project.

*git branch –D branch\_name*

*remotes*. A remote is a shared Git repository that allows multiple collaborators to work on the same Git project from different locations.

Collaborators work on the project independently, and merge changes together when they are ready to do so.

1. create a replica of Git suppository

*git clone remote\_location clone\_name*

* remote\_location tells Git where to go to find the remote. This could be a web address, or a filepath, such as:

/Users/teachers/Documents/some-remote

* clone\_name is the name you give to the **directory in which Git will clone** the repository.
* give the remote address the name *origin*,

1. see a list of a Git project's remotes

*git remote -v*

1. see if changes have been made to the remote and bring the changes down to your local copy

*git fetch*

brings those changes onto what's called a *remote branch*

1. push your branch up to the remote, origin.

*git push origin your\_branch\_name*

The workflow for Git collaborations typically follows this order:

1. Fetch and merge changes from the remote
2. Create a branch to work on a new project feature
3. Develop the feature on your branch and commit your work
4. Fetch and merge from the remote again (in case new commits were made while you were working)
5. *Push* your branch up to the remote for review

* Commands related to remote repository:
  + git clone git@github.com:USER-NAME/REPOSITORY-NAME.git or git clone https://github.com/user-name/repository-name.git
  + git push origin master
* Commands related to workflow:
  + git add .
  + git commit -m "A message describing what you have done to make this snapshot different"
* Commands related to checking status or log history
  + git status
  + git log

The git syntax works like this: program | action | destination.

For example:

* git add . is read like git | add | .
* git commit -m "message" is read like git | commit -m | "message"
* git status is read like git | status | (no destination)