GIT AND GITHUB

COMPENDIUM

WRITTEN BY

ANRICH TAIT

12 FEBRUARY 2023

Contents

1 SYNOPSIS

This compendium includes most of my knowledge on using Git and Github.

1.1 HOW TO USE THIS DOCUMENT

The dollar sign **$** indicates the start of a terminal command. Every now and then I will have so Arch linux commands sprinkled in so beware Windows users.

2 THE BASICS

2.1 WHAT IS GIT?

Git is a popular version control system. It was created by Linus Torvalds in 2005 and has been maintained by Junio Hamano since then. It is used for:

- Tracking code changes
- Tracking who made changes
- Coding collaboration

2.2 WHAT DOES GIT DO?

- Manage projects with repositories
- Clone a project to work on a local copy
- Control and track changes with Staging and Committing
- Branch and Merge to allow for work on different parts and versions of a project
- Pull the latest version of the project to a local copy
- Push local updates to the main project

2.3 WORKING WITH GIT

- Initiliaze Git on a folder, making it a Repository
- Git now creates a hidden folder to keep track of changes in the folder
- When a file is changed, added or deleted, it is considered modified
- You select the modified files you want to Stage
- The Staged files are Commited, which prompts Git to store a permanent snapshot of the files
- Git allows you to see the full history of every commit
- You can revert back to any previous commit
- Git does not store a seperate copy of every file in every commit, but keeps track of changes made in each commit

2.4 WHY GIT?

- Over 70% of developers use Git
- Developers can work together from anywhere in the world
- Developers can see the full history of the project
- Developers can revert to earlier versions of a project

2.5 WHAT IS GITHUB?

- Git is not the same as GitHub
- GitHub makes tools that use Git
- GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018.
- This document mainly focuses on integration between Git and Github

3 GETTING STARTED WITH GIT

3.1 GIT INSTALL

-To download Git use the following link: `https://git-scm.com/`

- For my fellow Arch users use the following link: `https://wiki.archlinux.org/title/git`

3.2 USING GIT IN THE COMMAND LINE

To start using Git open the command shell of your choice (I use Alacritty).
First check if git is properly installed: $ git –version = git version 2.39.1
If git is installed it will output a version like shown above.

3.3 CONFIGURE GIT

Enter your personal information, this will be important for later when integrating with GitHub. Each Git commit will use this information: $ git config –global user.name "Your Name" $ git config –gloabal user.email "your.email@gmail.com"

Note: Use "global" to set the username and email for **every repository** on your computer. If you want to set the username/email for just the current repo, you can remove "global

3.4 CREATING GIT FOLDER

Create a folder for your project: $ mkdir testproject (mkdir = make directory) $ cd testproject (cd = change directory)

Note: If you already have a directory you would like to use for Git: Navigate to it in the command line, or open it in your file explorer, right-click and select "Git Bash here"

3.5 INITIALIZE GIT

Once you have navigated to the correct folder you can initialize Git on that folder: $ git init = Initiliazed empty Git repository in / Users / user / myproject /.git /

Note: Git now knows that it should watch the folder you initiated it on. Git creates a hidden folder to keep track of changes.

4 GIT NEW FILES

4.1 ADDING NEW FILES IN GIT

You just created your first local Git repo, but it is empty. So let's add some files, or create a new file using your favorite text editor, then save or move it to the folder you just created.

I use emacs for my text editor so this is usually how I create new files. As an example I will create a basic C++ file:

1. In terminal: $ emacs testproject.cpp

2. The above command will create and open the testproject.cpp file. For the example I entered the following:

#include "testproject.h" #include <iostream> #include <string> using namespace std;

int main() { cout «"THIS IS A TEST FILE"; return 0; }

You can then go back to the terminal and list the files in your current working directory: $ ls (ls = list files/folders in directory) = testproject.cpp

We can then check the Git **status** and see if it is part of our repo: $ git status = On branch master

No commits yet

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

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

So now Git is **aware** of the file but has not added it to the repository yet. Files in your Git repository folder can be in one of 2 states:

- Tracked = files that Git knows about and are added to the repository
- Untracked = files that are in your working directory, but not yet added to the repository

When you first add files to an empty repository, they are all untracked. To get Git to track them you need to stage them, or add them to the staging enviroment.

5 GIT STAGING ENVIROMENT

One of the core functions of Git is the concepts around the **Staging Enviroment** and the **Commit.**

As you work on and edit a project you you will reach milestones or complete a section. This is where the **Staging Enviroment** comes in. **Staged** files are files that are ready to be comitted to the repository you are working on.

Return to your test document and add it to the Staging Enviroment: $git add testproject.cpp

Now check the status: $ git status testproject.cpp = On branch master

No commits yet

Changes to be committed: (use "git rm –cached <file>. . . " to unstage) new file: testproject.cpp

Check for a similar output to confirm that the file is indeed in the Staging Enviroment.

5.1 ADD MORE THAN ONE FILE

You can also stage more than one file at a time. Add 2 more files to your working folder using the text editor of your choice. For example I added a "README.md" file that describes the repository (This is recommended for all repositories) as well as a super basic python file. So now if you type "ls -al" you should see something like this: rwxr-xr-x 3 anricht anricht 4096 Feb 16 18:55 . drwxr-xr-x 4 anricht anricht 4096 Feb 16 15:00 .. drwxr-xr-x 7 anricht anricht 4096 Feb 16 18:44 .git -rw-r–r– 1 anricht anricht 149 Feb 16

18:50 README.md <————————-README file -rw-r–r– 1 anricht anricht 148 Feb 14 18:20 testproject.cpp <————————-C++ file -rw-r–r– 1 anricht anricht 77 Feb 16 18:55 testproject.py <————————-Python file

Obviously when your projects get bigger adding each file one at a time won't be very productive. So instead use the following command to add all files in your current directory to the Staging Enviroment: $ git add –all

Using –all instead of individual filenames will stage all changes (new,modified and deleted) files. Now if you type "git status" you should see something like this: $ git status = On branch master

No commits yet

Changes to be committed: (use "git rm –cached <file>..." to unstage) new file: README.md new file: testproject.cpp new file: testproject.py

This shows that all files have been added to the Staging Enviroment. **Note**: The short command for git add –all is = git add -A

6 GIT COMMIT

When you are finished editing the necessary files you are ready to move from the staging enviroment to the commit stage.

Adding commits keeps track of your progress and changes as you work. Git considers each commit change point or "save point". It is a point in the project you can go back to if you find a bug or want to make a change.

When you commit you should always include a message. By adding clear messages to each commit it is easy for yourself and others to see what has changed and when. **$ git commit -m "First release of the Git compendium to the repository!"** = [master (root-commit) f4d414d] First release of the Git compendium to the repository! 3 files changed, 15 insertions(+) create mode 100644 README.md create mode 100644 testproject.cpp create mode 100644 testproject.py

The commit command commits all files in the Staging Enviroment to the repository and the -m "message" commands adds a message.

7 Important links:

bibliographystyle:apacite bibliography:/path/to/library.bib