GIT & GitHub

-- By Fusion

Links

GIT Basic Commands
https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html

GIT Software Downloads
https://git-scm.com/downloads

GIT HUB Account Creation
https://github.com/

What is GIT

- Git is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency

- Git relies on the basis of distributed development of software where more than one developer may have access to the source code of a specific application and can modify changes to it that may be seen by other developers..

- Git allows a team of people to work together, all using the same files. It helps the team cope with the confusion that tends to happen when multiple people are editing the same files.

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

Features of GIT

- 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 Committed, 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 separate copy of every file in every commit, but keeps track of changes made in each commit!

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
- GitHub allows developers to create and manage the code in the repository in the remote location where others can aceses the code or Github is an collection repositories which contains the files of the project.

What is GitHub Repository

- GitHub Repository is an repo where all the files of an project will be stored and entire history of the code will also be maintained.

Difference between Git & GitHub

	GIT	GitHub
1	Git is a software	GitHub is a service.
2	Git is a command-line tool	GitHub is a graphical user interface
3	Git is installed locally on the system	GitHub is hosted on the web
4	Git is maintained by linux	GitHub is maintained by Microsoft.
5	Git is focused on version control and code sharing.	GitHub is focused on centralized source code hosting.

PUSH

The process of sending Code from eclipse to GIT local repository daily is called as Commit. And The process of sending Code from GIT local repository to remote GIT HUB repository is called as Push.(check out)

Git Commands for Push operation

Configuring git for the first time :-

```
git config --global user.name "yourname"
git config --global user.email "your email“
```

Status of Files

```
git status
```

Git Commands for Push operation

Create a New local Repository

Git creates a hidden folder to keep track of changes

Git init

Add Files

Staged files are files that are ready to be committed to the repository you are working on. 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 environment

Git add <filename with extension>

Git add *

Commit

Commit changes to head (but not yet to the remote repository):

Git commit –m “Commit message”

Git Commands for Push operation

Connects to a remote repository

Copy the url or the link of the repository. ‘git remote add origin ’ specifies that we are adding a remote repository, with the specified URL, as an origin to our local Git repo. Finally, pushing our master branch to the origin URL (remote repo) and set it as the default remote branch

git remote add origin <server>

Push the branch to your remote repository, So others can use it

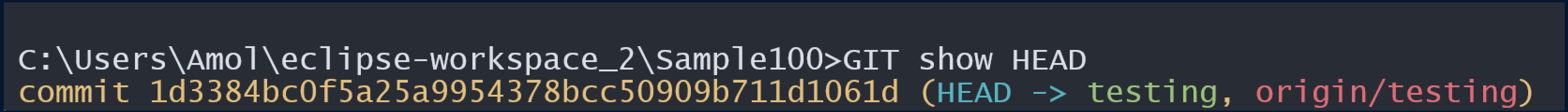
git push origin <branchname>

Git HEAD

The **HEAD** points out the last commit in the current checkout branch. It is like a pointer to any reference.

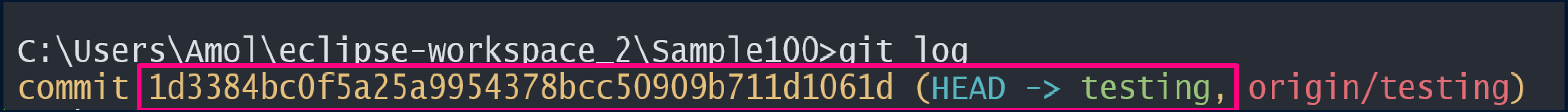
The **git show head** is used to check the status of the Head. This command will show the location of the Head.

git show HEAD



```
C:\Users\Amol\eclipse-workspace_2\Sample100>GIT show HEAD
commit 1d3384bc0f5a25a9954378bcc50909b711d1061d (HEAD -> testing, origin/testing)
```

In the above output, you can see that the commit id for the Head is given. It means the Head is on the given commit. Now, check the commit history of the project. You can use the git log command to check the commit history. See the below output:



```
C:\Users\Amol\eclipse-workspace_2\Sample100>git log
commit 1d3384bc0f5a25a9954378bcc50909b711d1061d (HEAD -> testing, origin/testing)
```

Log of a File

Git log command is one of the most usual commands of git. It is the most useful command for Git. Every time you need to check the history, you have to use the git log command. The basic git log command will display the most recent commits and the status of the head.

```
git log
```

Display the output as one commit per line:

```
git log –oneline
```

Displays the files that have been modified:

```
git log –stat
```

Display the modified files with location:

```
git log -p
```

Log of a File

Git log command allows viewing your git log as a graph. To list the commits in the form of a graph, run the git log command with --graph option. It will run as follows:

git log --graph

To make the output more specific, you can combine this command with --oneline option. It will operate as follows:

git log –graph –oneline

Filtering the commit history

By Amount:

To limit the git log's output, including the -<n> option. If we want only the last three commit, then we can pass the argument -3 in the git log command

Git log -3

Filtering the commit history

By date and time

We can filter the output by date and time. We have to pass **--after** or **-before** argument to specify the date. These both argument accept a variety of date formats. It will run as follows:

```
git log –after "yy-mm-dd"
```

We can also track the commits between two dates. To track the commits that were created between two dates, pass a statement reference **--before** and **--after** the date. Suppose, we want to track the commits between "2023-11-01" and "2023-11-08". We will run the command as follows:

```
git log –after= "2023-11-01" –before= "2023-11-08"
```

By Author

```
git log –author= "Author Name"
```

By Commit Message

```
git log –grep= "Commit Message"
```

Git Help

If you are having trouble remembering commands or options for commands, you can use Git help

See all the available options for the specific command

```
git –help
```

See all possible commands –

```
git help --all
```

If you find yourself stuck in the list view, SHIFT + G to jump the end of the list, then q to exit the view

Git clone from GitHub :-

We can clone repository from GitHub

```
git clone <copied URL>
```

Git clone branch

Git allows making a copy of only a particular branch from a repository. You can make a directory for the individual branch by using the git clone command. To make a clone branch, you need to specify the branch name with -b command. Below is the syntax of the command to clone the specific git branch:

```
git clone -b <Branch_Name> <Repository URL >
```

Pull

The process of extracting Code from remote GIT HUB repository to GIT local repository to is called as Pull.

Command

Git pull origin <branchName>

Git Branching

In Git, a branch is a new/separate version of the main repository. Branches allow you to work on different parts of a project without impacting the main branch. When the work is complete, a branch can be merged with the main project. We can even switch between branches and work on different projects without them interfering with each other

Making a New Git Branch

```
git branch <name of branch>
```

Checking all available Branches

```
git branch
```

Switching to other branches

```
git checkout <branch_name>
```

Making a New branch and directly switching to it

```
git checkout –b <branch_name>
```

Deleting a branch

```
git branch –d <branch name>
```

Merging two branches

It’s preferred to change/switch to master branch before any branch needs to be merged with it.

```
git merge <branch name>
```

This will merge the specified branch with our master branch.

What is a Git Merge Conflict?

A git merge conflict is an event that takes place when Git is unable to automatically resolve differences in code between two commits. Git can merge the changes automatically only if the commits are on different lines or branches.

Let’s assume there are two developers: Developer A and Developer B. Both of them pull the same code file from the remote repository and try to make various amendments in that file. After making the changes, Developer A pushes the file back to the remote repository from his local repository. Now, when Developer B tries to push that file after making the changes from his end, he is unable to do so, as the file has already been changed in the remote repository.
To prevent such conflicts, developers work in separate isolated branches. The Git merge command combines separate branches and resolves any conflicting edits.

How to resolve Merge Conflict in GIT

There are a few steps that could reduce the steps needed to resolve merge conflicts in Git.
Step 1: The easiest way to resolve a conflicted file is to open it and make any necessary changes.

Step 2: After editing the file, we can use the git add a command to stage the new merged content.

Step 3: The final step is to create a new commit with the help of the git commit command.

Step 4: Git will create a new merge commit to finalize the merge.

How to resolve Merge Conflict in GIT

git log --merge

The git log --merge command helps to produce the list of commits that are causing the conflict.

git diff

The git diff command helps to identify the differences between the states repositories or files.

•

Undoing Changes

Case 1 Stages Changes

Git reset <FileName>
Git Reset

Case 2 Commited Changes [for latest commit]

Git reset HEAD~1

Case 3 Commited changes [for Many commit]
Git reset <Commit hash>
Git reset --hard <commit_hash>

Git Stash

Sometimes you want to switch the branches, but you are working on an incomplete part of your current project. You don't want to make a commit of half-done work. Git stashing allows you to do so. The **git stash command** enables you to switch branches without committing the current branch.

Generally, the stash's meaning is "**store something safely in a hidden place**.“Git temporarily saves your data safely without committing.

Some useful options are given below:

Stashes the modified files and creates new stash

git stash

If you want to give name for that particular stash

git stash save <name>

To see all the stashes list

```
git stash list
```

If you want to give name for that particular stash

```
git stash save <name>
```

To apply the particular stash

You can re-apply the changes that you just stashed by using the git stash command. To apply the commit, use the git stash command, followed by the apply option. It is used as:

```
git stash apply
```

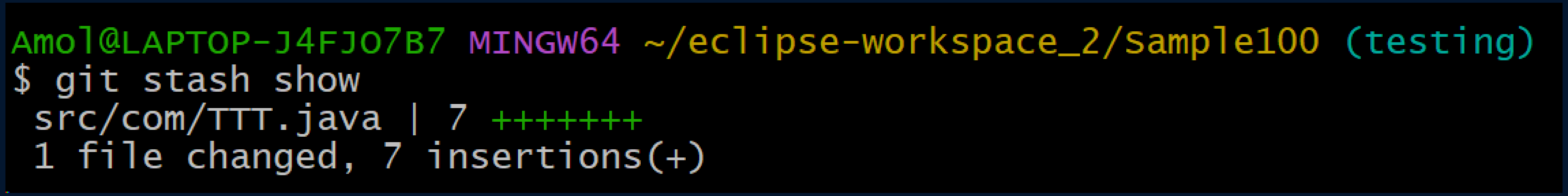
```
Git stash apply <stash-id>
Eg. git stash apply stash@{1}
```

If we don't specify a stash, Git takes the most recent stash and tries to apply it.

Git stash changes

We can track the stashes and their changes. To see the changes in the file before stash and after stash operation, run the below command:

git stash show

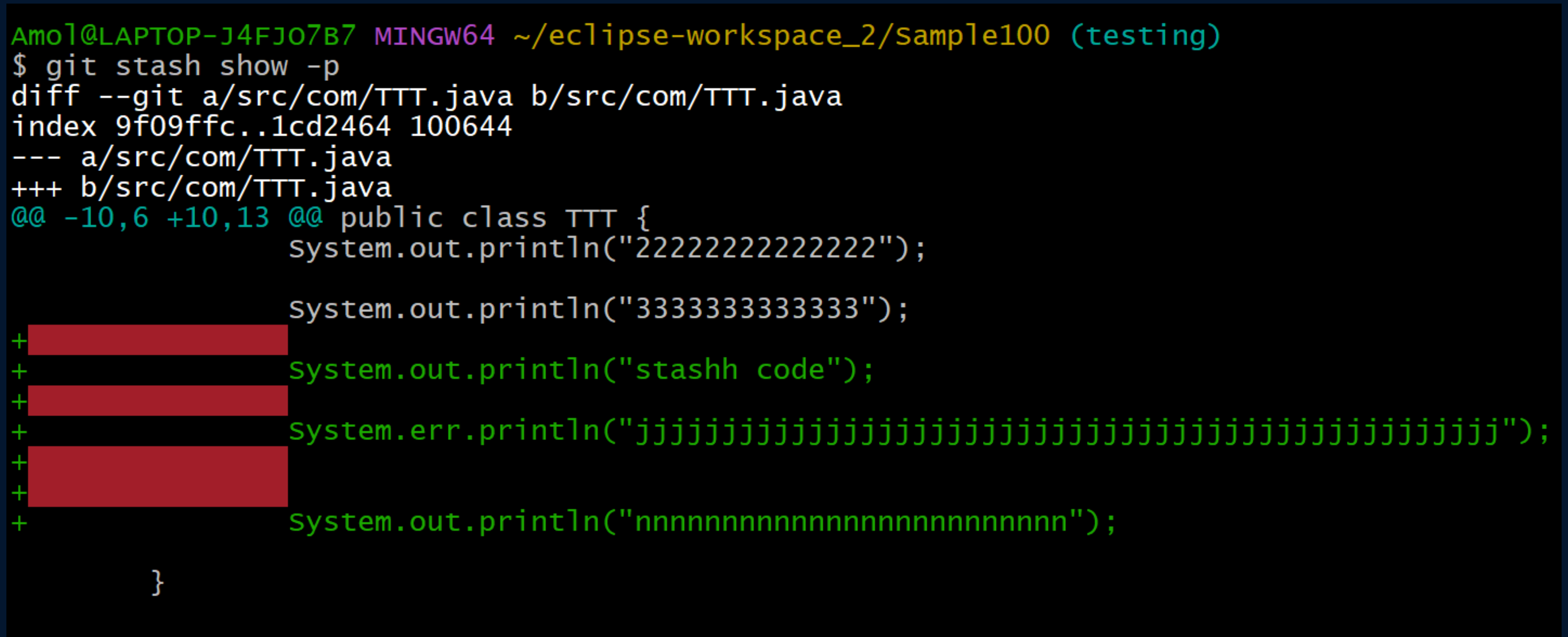


```
Amol@LAPTOP-J4FJO7B7 MINGW64 ~/eclipse-workspace_2/Sample100 (testing)
$ git stash show
 src/com/TTT.java | 7 +++++++
 1 file changed, 7 insertions(+)
```

The above output illustrates that there are two files that are stashed, and two insertions performed on them.

We can exactly track what changes are made on the file. To display the changed content of the file, perform the below command:

git stash show –p



```
Amol@LAPTOP-J4FJO7B7 MINGW64 ~/eclipse-workspace_2/Sample100 (testing)
$ git stash show -p
diff --git a/src/com/TTT.java b/src/com/TTT.java
index 9f09ffc..1cd2464 100644
--- a/src/com/TTT.java
+++ b/src/com/TTT.java
@@ -10,6 +10,13 @@ public class TTT {
                System.out.println("22222222222222");

                System.out.println("3333333333333");
+
+               System.out.println("stashh code");
+
+               System.err.println("jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj");
+
+
+               System.out.println("nnnnnnnnnnnnnnnnnnnnnnn");

        }
```

Git stash pop (Reapplying stash changes)

Git allows the user to re-apply the previous commits by using git stash pop command. The popping option removes the changes from stash and applies them to your working file.
The git stash pop command is quite similar to git stash apply. The main difference between both of these commands is stash pop command that deletes the stash from the stack after it is applied.

git stash pop

Git stash drop (Unstash)

The **git stash drop** command is used to delete a stash from the queue. Generally, it deletes the most recent stash. Caution should be taken before using stash drop command, as it is difficult to undo if once applied.
The only way to revert it is if you do not close the terminal after deleting the stash. The stash drop command will be used as:

git stash drop the most recent stash has been drop

To delete a particular stash from the available stashes, pass the stash id in stash drop command

Git stash drop <stash-id>

Git stash clear

The **git stash clear** command allows deleting all the available stashes at once. To delete all the available stashes, operate below command:

```
git stash clear
```

Git stash branch

If you stashed some work on a particular branch and continue working on that branch. Then, it may create a conflict during merging. So, it is good to stash work on a separate branch.

The git stash branch command allows the user to stash work on a separate branch to avoid conflicts. The syntax for this branch is as follows:

```
git stash branch <Branch name>
```

The above command will create a new branch and transfer the stashed work on that.

Thank You

Best of Luck