



Github Session 10

Summary 05-03-2022

- Suppose we have made multiple commits, each commit have its own history and at a point of time we will have a huge amount of histories stored including the unwanted histories also.
- If we want to take only relevant history while merging two branches, then **Squashing commits** allows you to create a cleaner and more concise commit history by combining multiple small, related commits into a single commit.
- When collaborating on a project, having a clean and focused commit history makes code reviews more manageable. Instead of reviewing numerous small commits, reviewers can focus on reviewing a smaller number of more meaningful commits, each representing a logical change or feature.
- To understand better we can see the example

The image is a composite of two screenshots. The left screenshot shows a Git GUI interface with a commit history graph. The graph shows a sequence of commits: 'dev1' (red box), 'ced0a42', 'ba85194', 'b959b8f' (green box), and 'fa606c5' (green box). A red box labeled 'HEAD -> master' points to the 'b959b8f' commit. Below 'fa606c5' are two more commits: 'ba5fcdc' and '490f5b1' (green box). The right screenshot shows a terminal window with the following commands and output:

```
Vimal Daga@DESKTOP-3E1AGGT MINGW64  
s-workshop/ws13 (master)  
$ cd ..  
cd  
Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitw  
s-workshop  
$ cd ws14  
Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitw  
s-workshop/ws14 (dev1)  
$ git checkout master  
Switched to branch 'master'  
Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitw  
s-workshop/ws14 (master)  
$ git log
```

- Here we can see that the master branch has 3 commits and the dev branch have other 3 new commits
- Now we want to merge them without taking the unwanted history, so for that we will use the command ***git merge --squash dev1***

The screenshot shows the Git GUI on the left and a terminal window on the right. The GUI displays a commit graph where the 'dev1' branch (red) has three commits (ced0a42, ba85194, b959b8f) and the 'master' branch (green) has three commits (fa606c5, ba5fcdc, 490f5b1). A red box labeled 'HEAD -> master' points to the 'master' branch. The terminal window shows the following commands and output:

```
Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitw
s-workshop/ws14 (master)
$ ls
a b c d

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitw
s-workshop/ws14 (master)
$ git merge --squash dev1
Updating fa606c5..ced0a42
Fast-forward
Squash commit -- not updating HEAD
d1 | 0
d2 | 0
d3 | 0
3 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 d1
create mode 100644 d2
create mode 100644 d3

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitw
s-workshop/ws14 (master)
$ ls
a b c d d1 d2 d3

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitw
s-workshop/ws14 (master)
$
```

- We can see that initially the master branch had only 3 commits(a b c d) and after merging we have the dev branch commits also in the master branch(a b c d d1 d2 d3).
- All of the dev branch changes combines and make a single commit and will merge to the master branch when we do squashing.

The screenshot shows the Git GUI on the left and a terminal window on the right. The GUI displays a commit graph where the 'dev1' branch (red) has three commits (ced0a42, ba85194, b959b8f) and the 'master' branch (green) has four commits (fa606c5, ba5fcdc, 490f5b1, 34c3764). A red box labeled 'HEAD -> master' points to the 'master' branch. The terminal window shows the following commands and output:

```
Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitw
s-workshop/ws14 (master)
$ git status -s

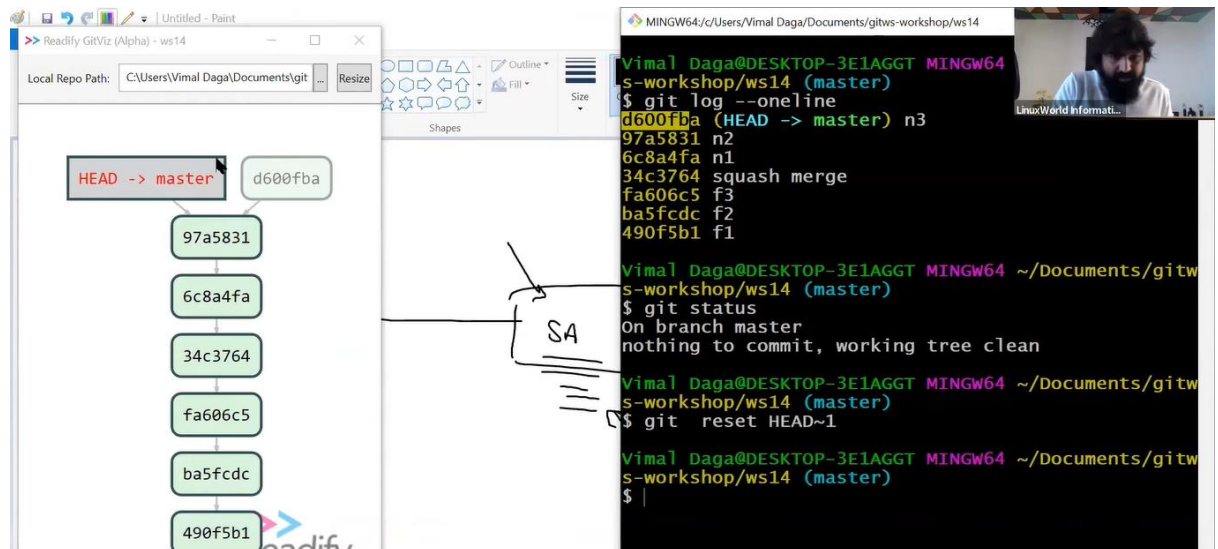
Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitw
s-workshop/ws14 (master)
$ ls
a b c d d1 d2 d3

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitw
s-workshop/ws14 (master)
$ git log --oneline
34c3764 (HEAD -> master) squash merge
fa606c5 f3
ba5fcdc f2
490f5b1 f1

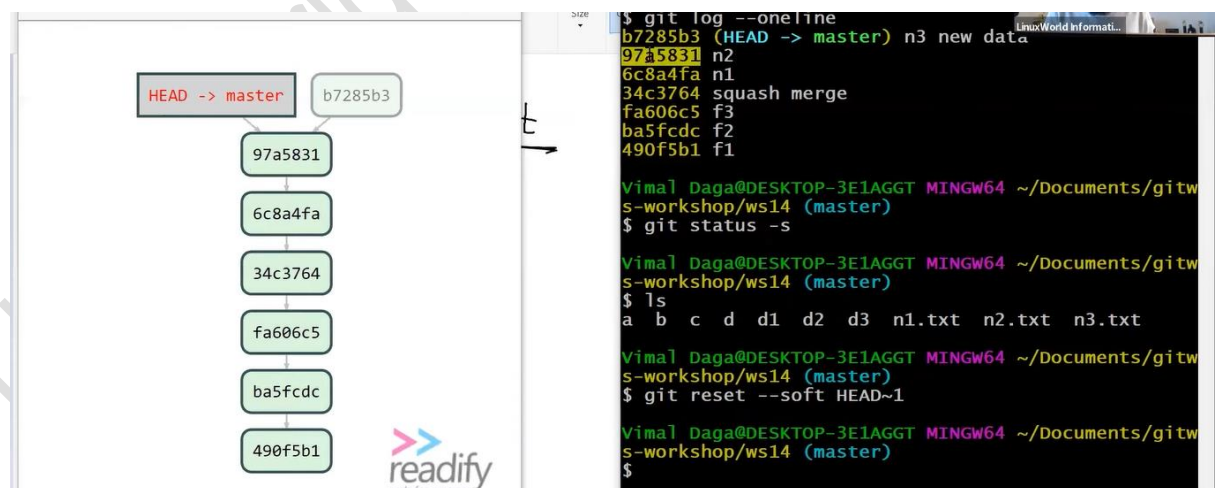
Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitw
s-workshop/ws14 (master)
$ git branch -d dev1
error: The branch 'dev1' is not fully merged.
If you are sure you want to delete it, run 'git bra
nch -D dev1'.

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitw
s-workshop/ws14 (master)
$
```

- To take back a file from the commit area to the staging area we use the **reset** for it ***git reset HEAD~1***
 - This command will remove the last commit from the commit area and will take it to the working area.



- A mixed reset is the default behaviour when no option is specified. It moves the HEAD to the specified commit and updates the staging area to match the state of the specified commit, but does not change the working directory.
- A soft reset moves the HEAD to the specified commit, leaving your working directory and index (staging area) unchanged. This means that the changes in your working directory and staging area remain intact, but the commits are "uncommitted" and moved to the staging area. For soft reset use the ***git reset --soft HEAD~1***



- A hard reset moves the HEAD to the specified commit and resets both the staging area and the working directory to match the state of the specified commit.
- The hard reset removes everything from the working area also.

The screenshot shows a Windows desktop with two windows. On the left is a 'Readify GitViz (Alpha) - ws14' window displaying a commit history diagram. The diagram shows a vertical chain of commits: 97a5831 (HEAD -> master), 6c8a4fa, 34c3764, fa606c5, ba5fcdc, and 490f5b1. A branch 'dbbd46d' is shown as an orphaned commit branching off from 97a5831. On the right is a 'MINGW64/c/Users/Vimal Daga/Documents/gitws-workshop/ws14' terminal window. The terminal shows the following commands and output:

```
Vimal Daga@DESKTOP-3E1AGGT MINGW64
s-workshop/ws14 (master)
$ git status -s
M n3.txt

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitw
s-workshop/ws14 (master)
$ git log --oneline
dbbd46d (HEAD -> master) n3 add after soft reset
97a5831 n2
6c8a4fa n1
34c3764 squash merge
fa606c5 f3
ba5fcdc f2
490f5b1 f1

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitw
s-workshop/ws14 (master)
$ git reset --hard HEAD~1
HEAD is now at 97a5831 n2

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitw
s-workshop/ws14 (master)
$
```

- The orphaned commit is removed from the timeline but git internally keeps it for a safe side and we can restore it back.

➤ To restore back the deleted or the orphaned commit, first move the HEAD to that commit then create a new branch at that head.

The screenshot shows the same Windows desktop as before. The 'Readify GitViz' window on the left is identical, showing the commit history with the orphaned 'dbbd46d' branch. The terminal window on the right shows the following commands and output:

```
Vimal Daga@DESKTOP-3E1AGGT MINGW64
s-workshop/ws14 (master)
$ git checkout dbbd46d
Note: switching to 'dbbd46d'.

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 switching back to a branch.

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

  git switch -c <new-branch-name>

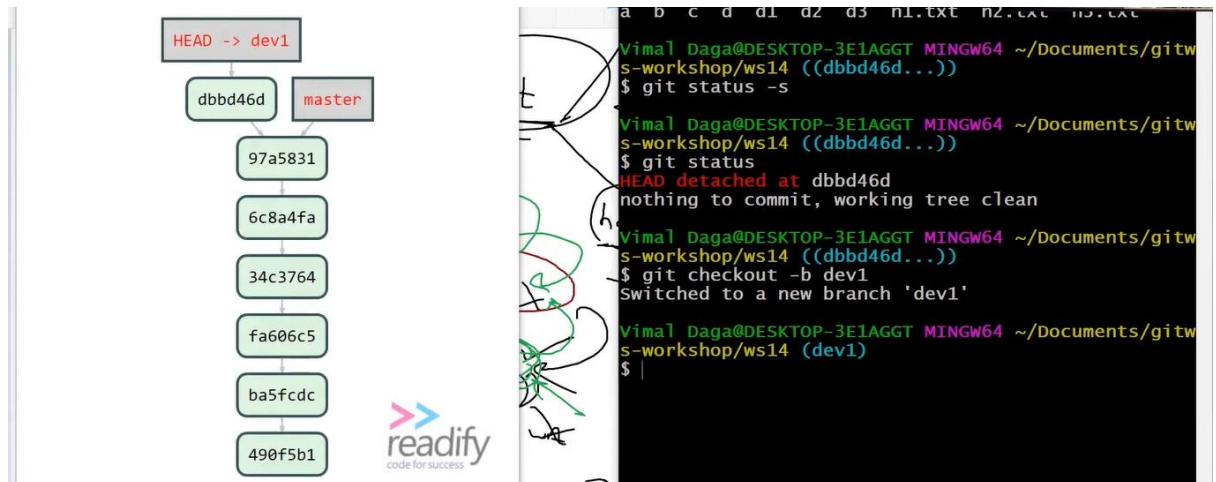
Or undo this operation with:

  git switch -

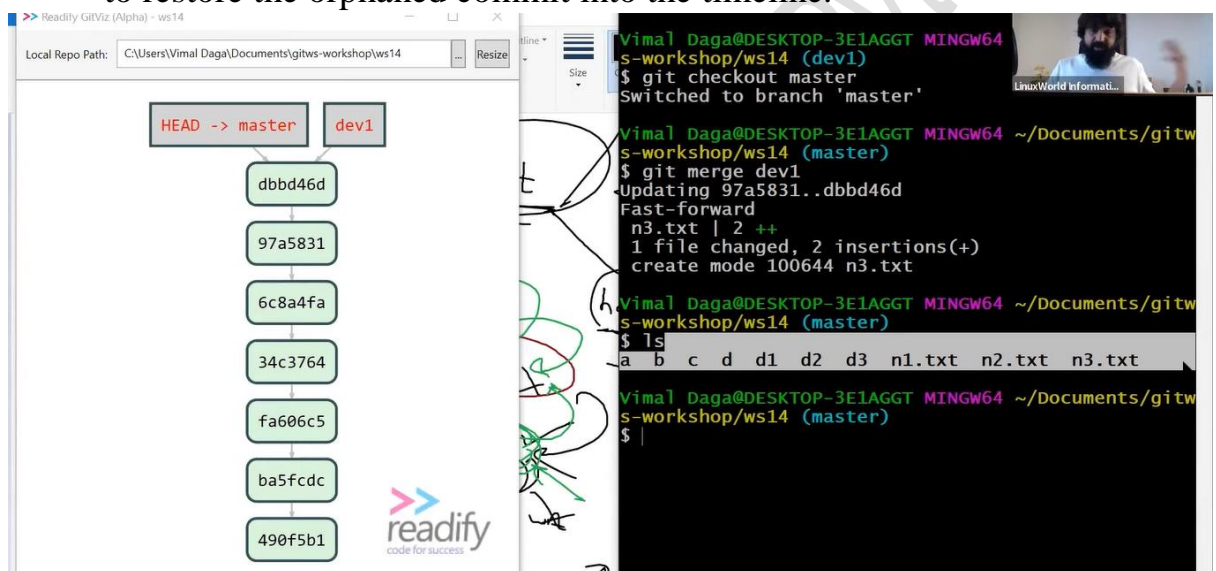
Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at dbbd46d n3 add after soft reset

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitw
s-workshop/ws14 ((dbbd46d...))
$
```

- Once the new branch is created, merge it back to the master branch to restore the orphaned commit into the timeline.



- Whenever we do some changes in the github directly, we doesn't get updated in the local system or we can say that local system will not get to know about any changes done on the github.
- To deal with this problem or to get information about any changes that happened in the github we can use the **fetch** for it. *git fetch* .
- Fetch will give you only the information about any update or the change but will not give you the data.

[Mastering Git and Github]

```
Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitws-workshop/ws15 (master)
$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 640 bytes | 21.00 KiB/s, done.
From github.com:vimallinuxworld13/gittrainingtest111
6bc9388..4131344 master -> origin/master

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitws-workshop/ws15 (master)
$ git status
On branch master
Your branch is behind 'origin/master' by 1 commit,
and can be fast-forwarded.
(use "git pull" to update your local branch)

nothing to commit, working tree clean

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitws-workshop/ws15 (master)
$
```

- To get the data also, we use the command **git pull** for it.

```
Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitws-workshop/ws15 (master)
$ cat a.txt
aa
bbb

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitws-workshop/ws15 (master)
$ git pull
Updating 6bc9388..4131344
Fast-forward
 a.txt | 1 +
 1 file changed, 1 insertion(+)

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitws-workshop/ws15 (master)
$
```

- Cloning a repository allows you to start working with a copy of the project, preserving its entire history and allowing you to make changes locally.
- To clone any repository we can use the command **git clone <repository url>**.

```
$ ls
git-workshop/  lwgit-ws/  ws1/  ws12/  ws14/  ws2/  ws4/

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitws-workshop
$ git clone https://github.com/vimallinuxworld13/gittrainingtest111.git
Cloning into 'gittrainingtest111'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 9 (delta 0), reused 6 (delta 0), pack-reused 0
Receiving objects: 100% (9/9), done.

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitws-workshop
$ ls
git-workshop/  lwgit-ws/  ws/  ws11/  ws13/  ws15/  ws3/  wsgit/
gittrainingtest111/  lwws/  ws1/  ws12/  ws14/  ws2/  ws4/

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitws-workshop
$ cd gittrainingtest111/

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/gitws-workshop/gittrainingtest111
(master)
$
```

- **git clone** is used to create a copy of an existing repository whereas **git pull** is used to update your local repository with changes from a remote repository.

Linux World Informatics Pvt Ltd