[CS200]-Stnt II **Fall 2020-21**

Homework: II

Group Name: The Medellin Cartel

1(a). If we want to work on multiple directories the there is a git command as:

```
$ git worktree add <path> <branch>
```

This command will simply help us to have multiple branches checked out at same time. Some other commands related to this are:

```
$ git worktree remove [-f] <worktree>
$ git worktree list [--porcelain]
$ git worktree lock [--reason <string>] <worktree>
$ git worktree add[-f][--detach][--checkout][--lock][-b<new-branch>]<path><commit-ish>
$ git worktree move <worktree> <new-path>
```

1(b). The commands below should be followed that lead to a directed edge from the Staging Area (Index) to the Working Directory:

```
$ echo "the medellin cartel" >> tmc
$ git add tmc
$ cat tmc
$ git commit -m "first commit"
$ echo "stnt II" >> tmc
$ cat tmc
$ git checkout -- tmc
$ cat tmc
```

The command:

```
$ git checkout -- tmc
```

overwrites the last staged version of the file ” tmc ” in the working directory. The other command that can be used for the same is

```
$ git restore tmc
```

We can use the following commands to overwrite the contents of tmc in the working directory with the blob which stored the data of the previously staged file.

```
1. $ git cat -file -p <hash code of previous version's blob> > tmc
2. $ git show <hash code of previous version's blob> > tmc
```

1(c). Command which we can use is:

```
$ git add -p <filename>
```

This command shows small portions of the changed files and then ask the user to take further step. In each step, you can mark hunks, which is a nearby set of changes, for staging or to be ignored for now.

1(d). We can use the command:

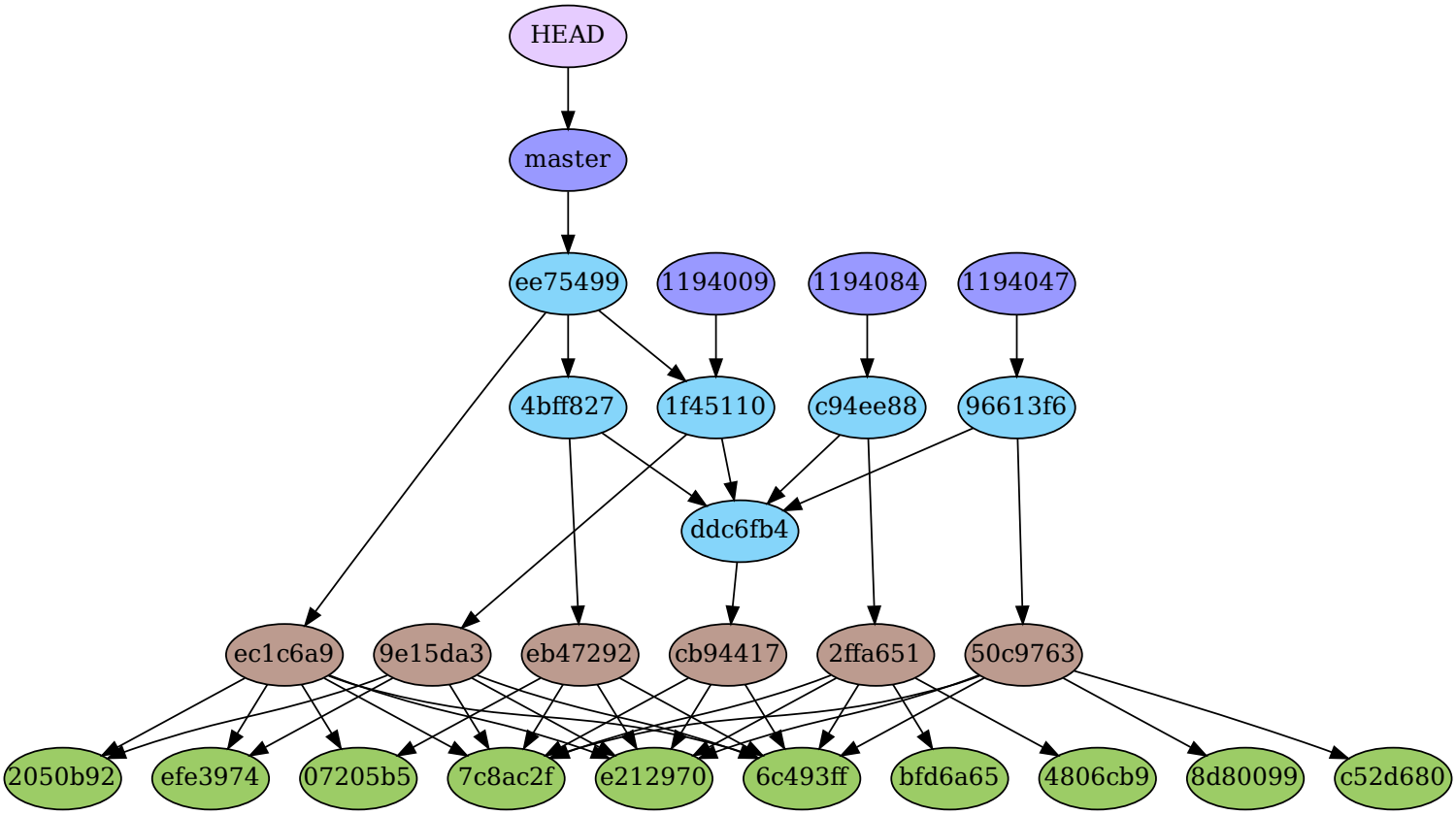
$\texttt{\$ git rebase -i HEAD} \sim N \quad [N \; is \; the \; number \; of \; commits \; you \; want \; to \; combine]$

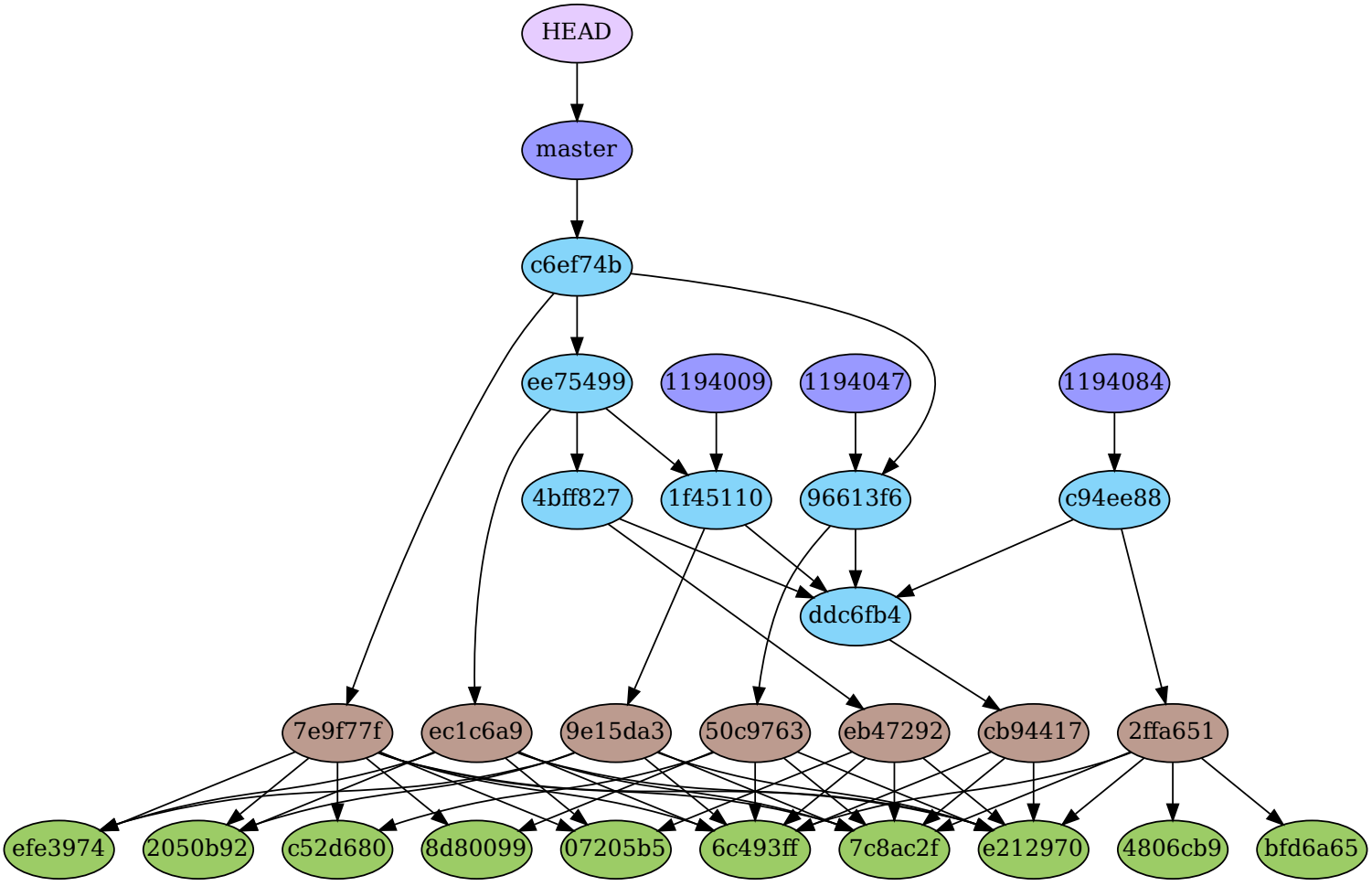
Git flow can be as the following:

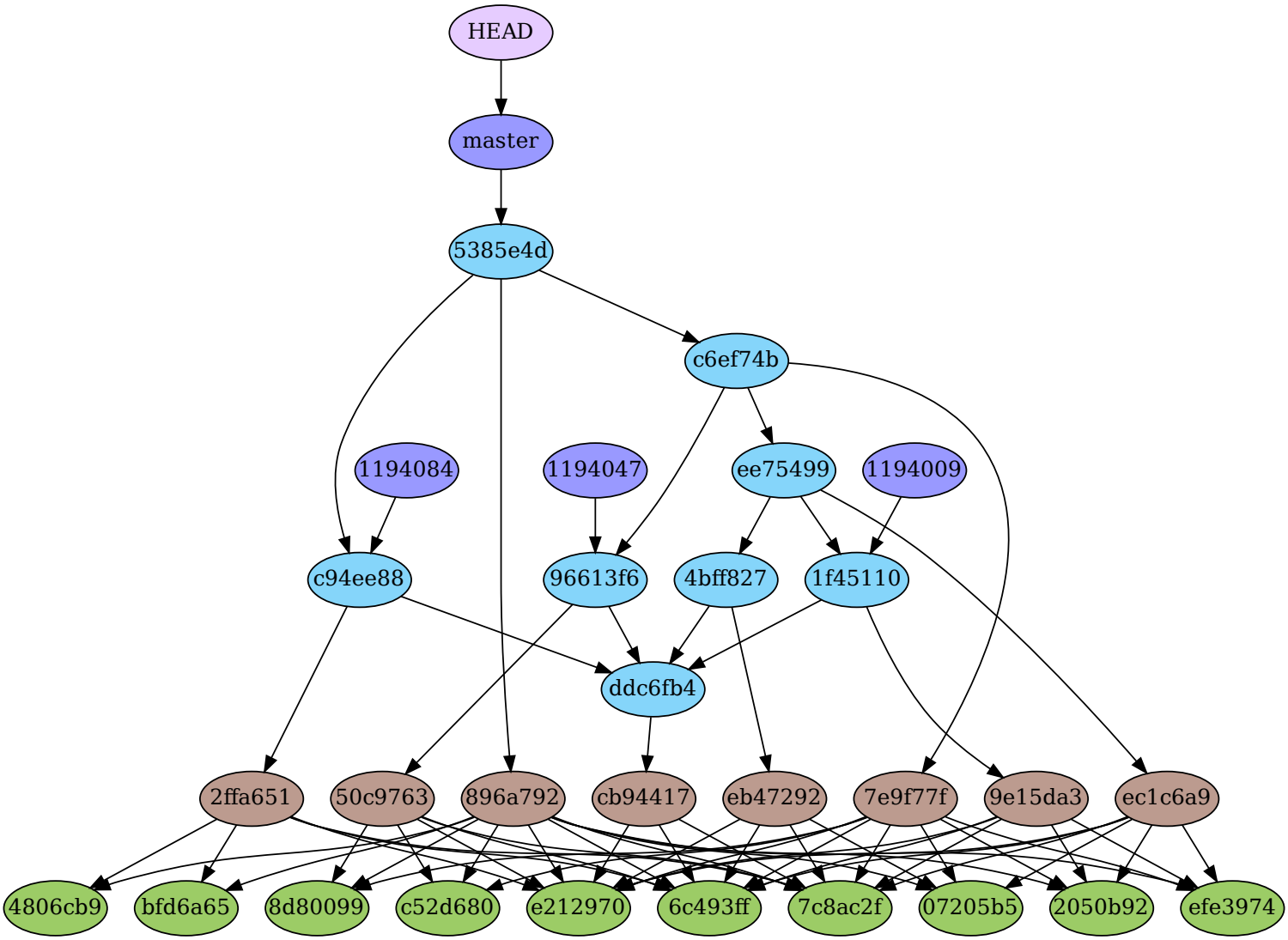
```
$ git init
$ touch file1.txt
$ echo "1" > file1.txt
$ git add file1.txt
$ git commit -m "first commit"
$ echo "2" > file1.txt
$ git add file1.txt
$ git commit -m "second commit"
$ echo "3" > file1.txt
$ git add file1.txt
$ git commit -m "third commit"
$ git graph
$ git rebase -i HEAD ~ 2
$ git graph
```

2(a). The shell script file named 2a.sh is present in the folder.
2(b). The shell script file 2b.sh is present in the folder and the following three pages shows merging in the given order:

```
1.11940090
2.11940470
3.11940840
```

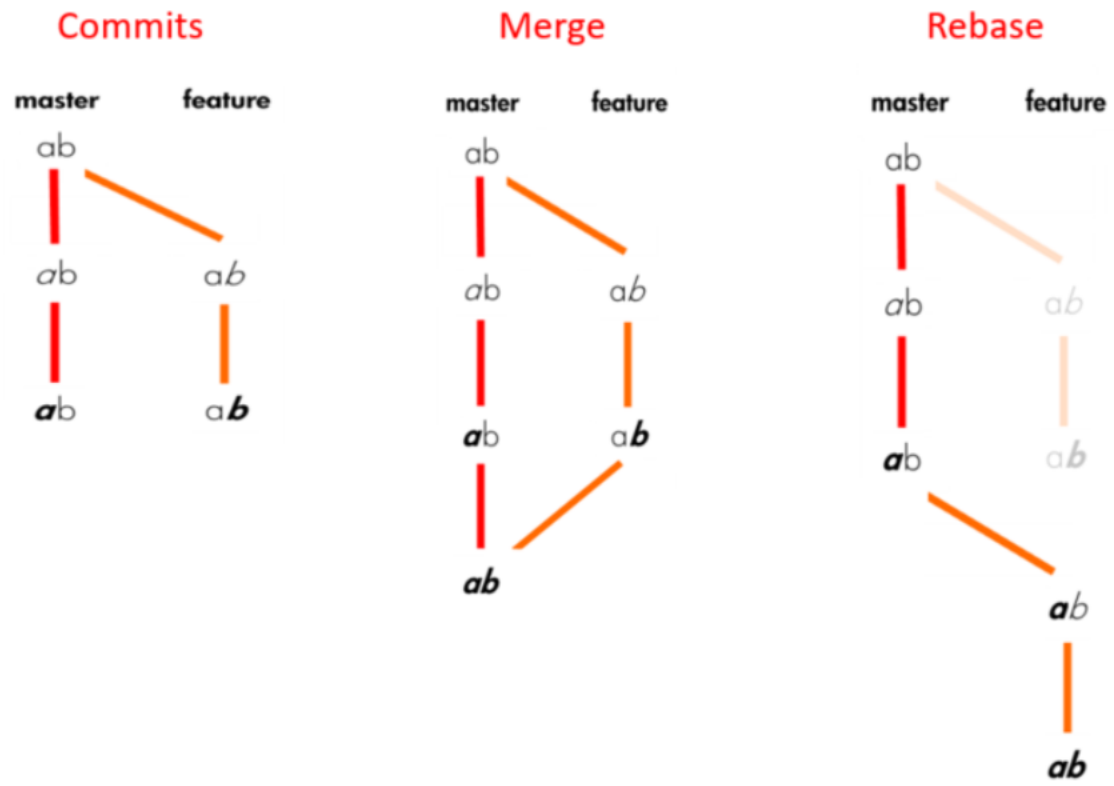

HEAD
master
ee75499
1194009
1194084
1194047
4bff827
1f45110
c94ee88
96613f6
ddc6fb4
ec1c6a9
9e15da3
eb47292
cb94417
2ffa651
50c9763
2050b92
efe3974
07205b5
7c8ac2f
e212970
6c493ff
bfd6a65
4806cb9
8d80099
c52d680


HEAD
master
c6ef74b
ee75499
1194009
1194047
1194084
4bff827
1f45110
96613f6
c94ee88
ddc6fb4
7e9f77f
ec1c6a9
9e15da3
50c9763
eb47292
cb94417
2ffa651
efe3974
2050b92
c52d680
8d80099
07205b5
6c493ff
7c8ac2f
e212970
4806cb9
bfd6a65


HEAD
master
5385e4d
c6ef74b
1194084
1194047
ee75499
1194009
c94ee88
96613f6
4bff827
1f45110
ddc6fb4
2ffa651
50c9763
896a792
cb94417
eb47292
7e9f77f
9e15da3
ec1c6a9
4806cb9
bfd6a65
8d80099
c52d680
e212970
6c493ff
7c8ac2f
07205b5
2050b92
efe3974

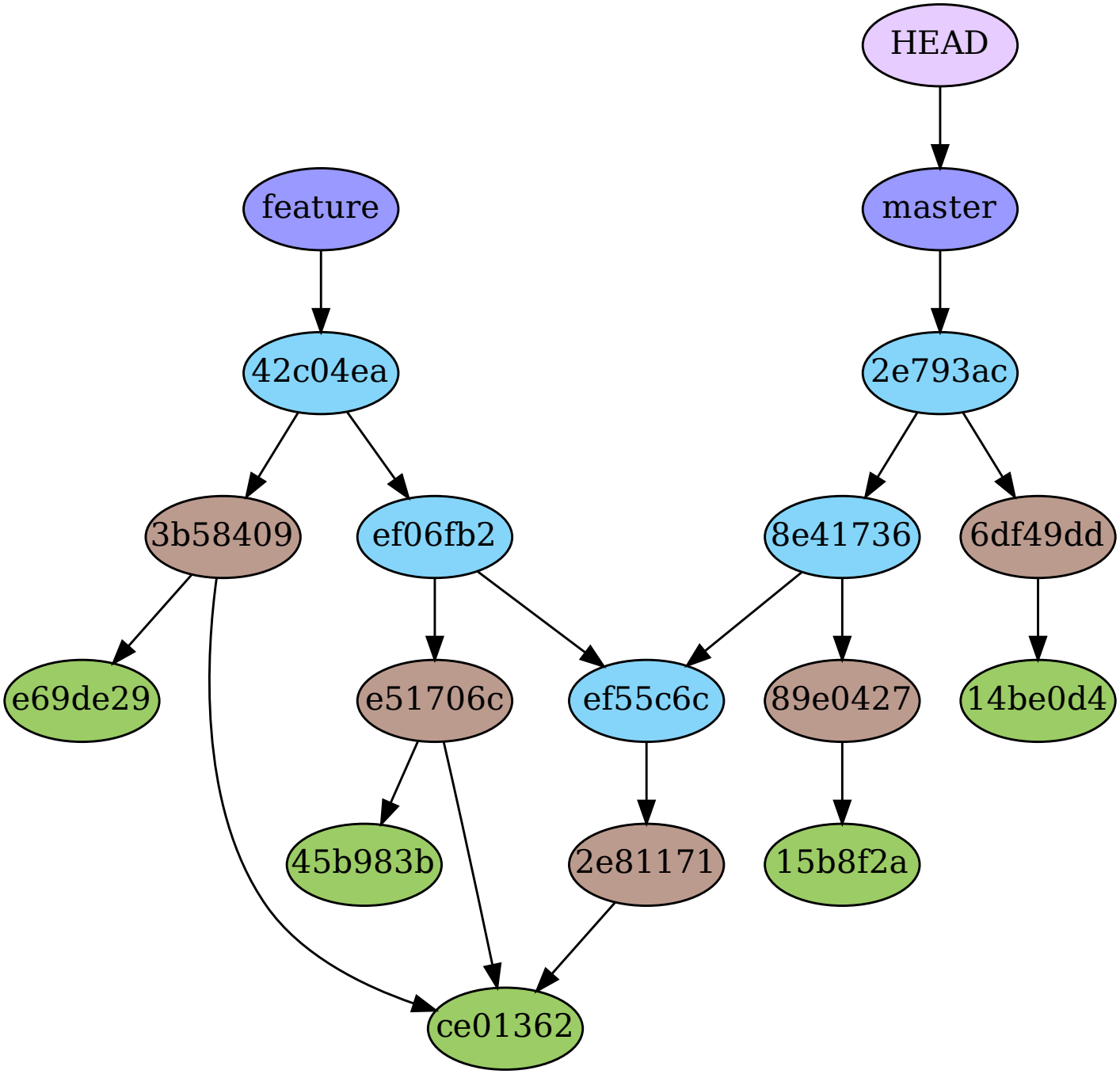
3(a). Case when Merge is better than Rebase:
This will be the case when we don't want to change the history of commits. Merging takes the contents of the feature branch and integrates it with the master branch. As a result of this the history of feature branch remains unchanged. So if we want to preserve the branch history we should opt for merge as rebase will rewrite the branch history by the point from where the branch was created.

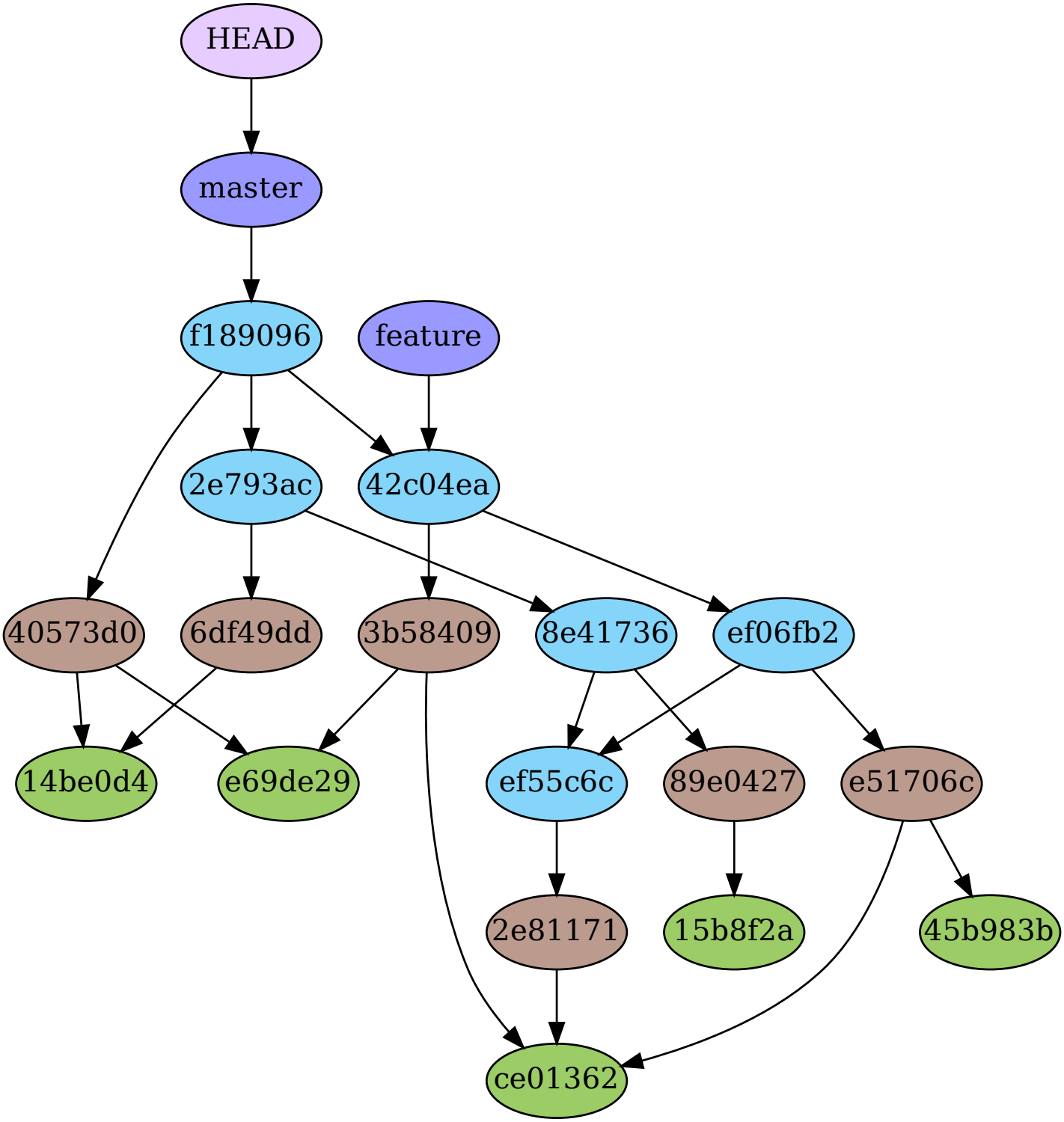
3(b). Case when Rebase is better than Merge:
This can be used when we don't have the need to preserve the history of the feature branch. We can rewrite it using rebase.

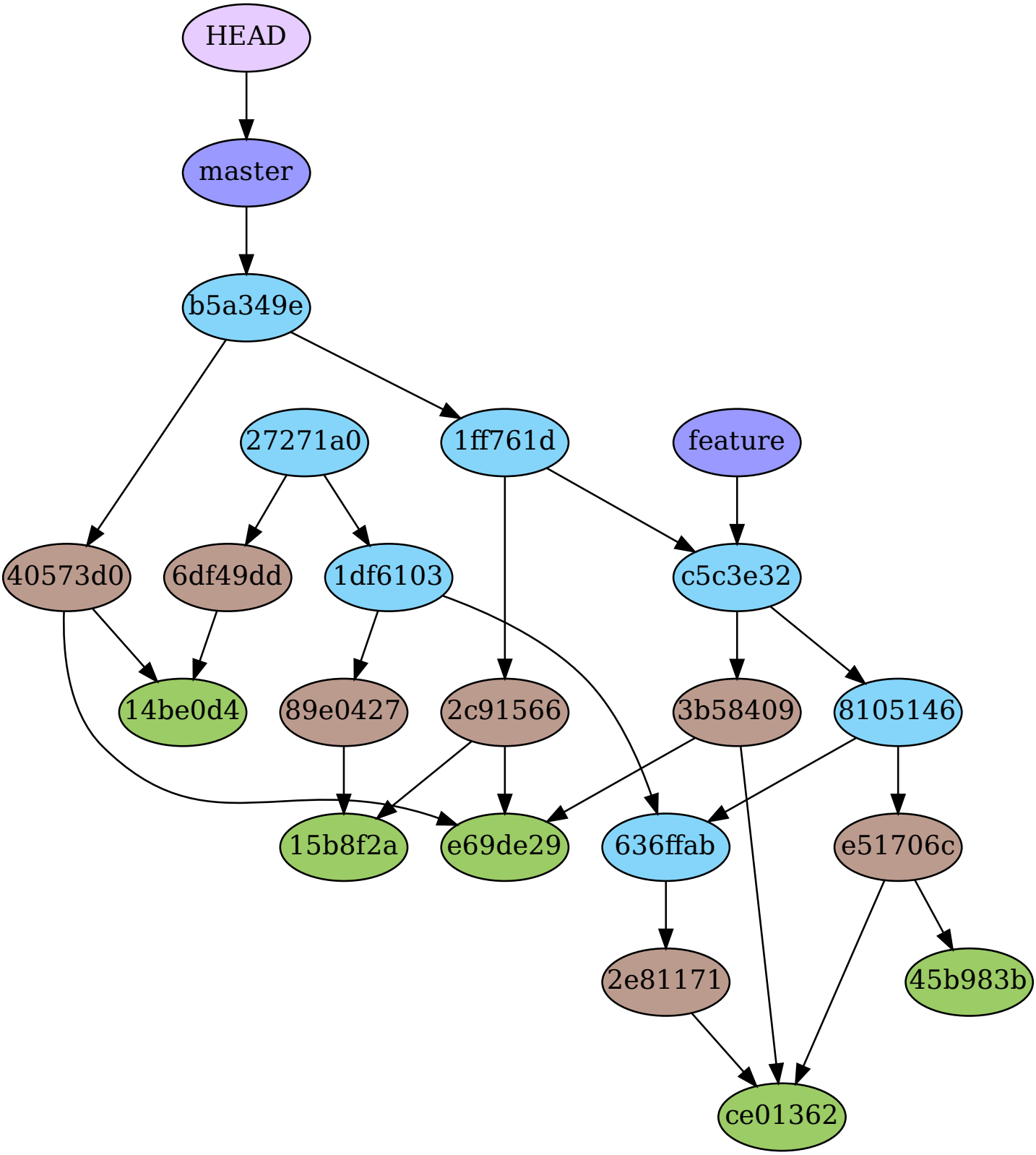


Following 3 git graphs are for:

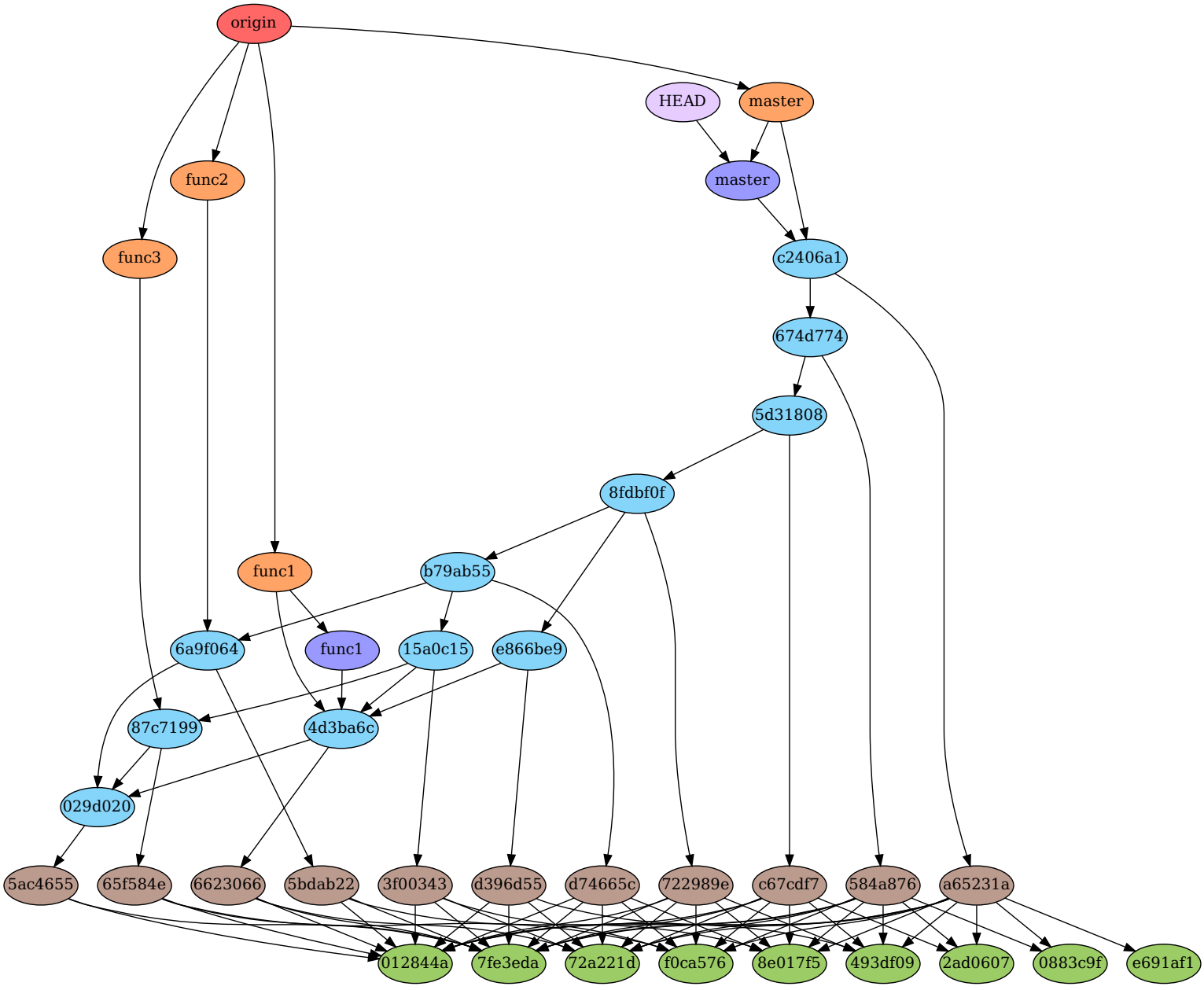
1.Commits
2.Merge
3.Rebase

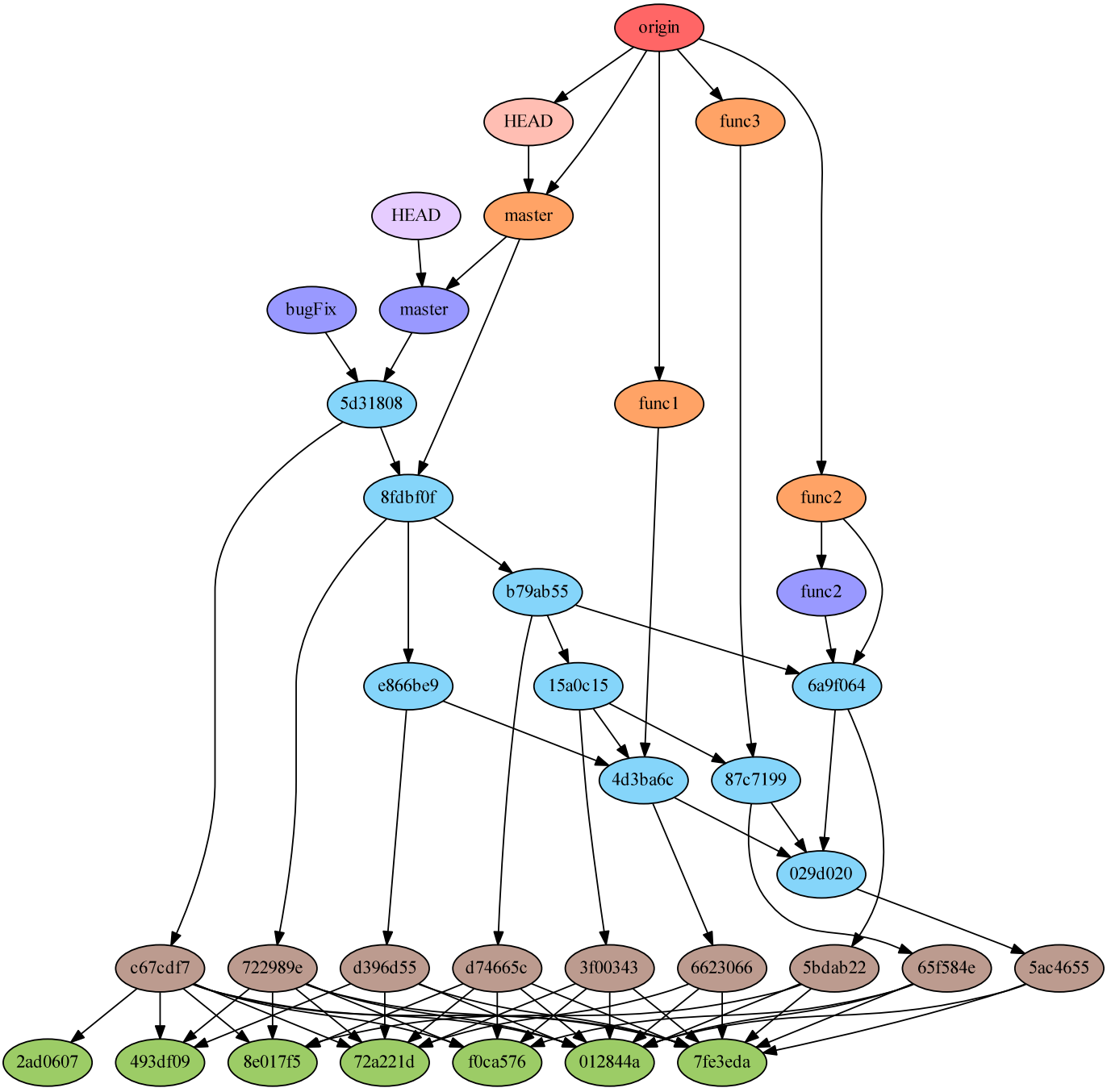

HEAD
feature
master
42c04ea
2e793ac
3b58409
ef06fb2
8e41736
6df49dd
e69de29
e51706c
ef55c6c
89e0427
14be0d4
45b983b
2e81171
15b8f2a
ce01362

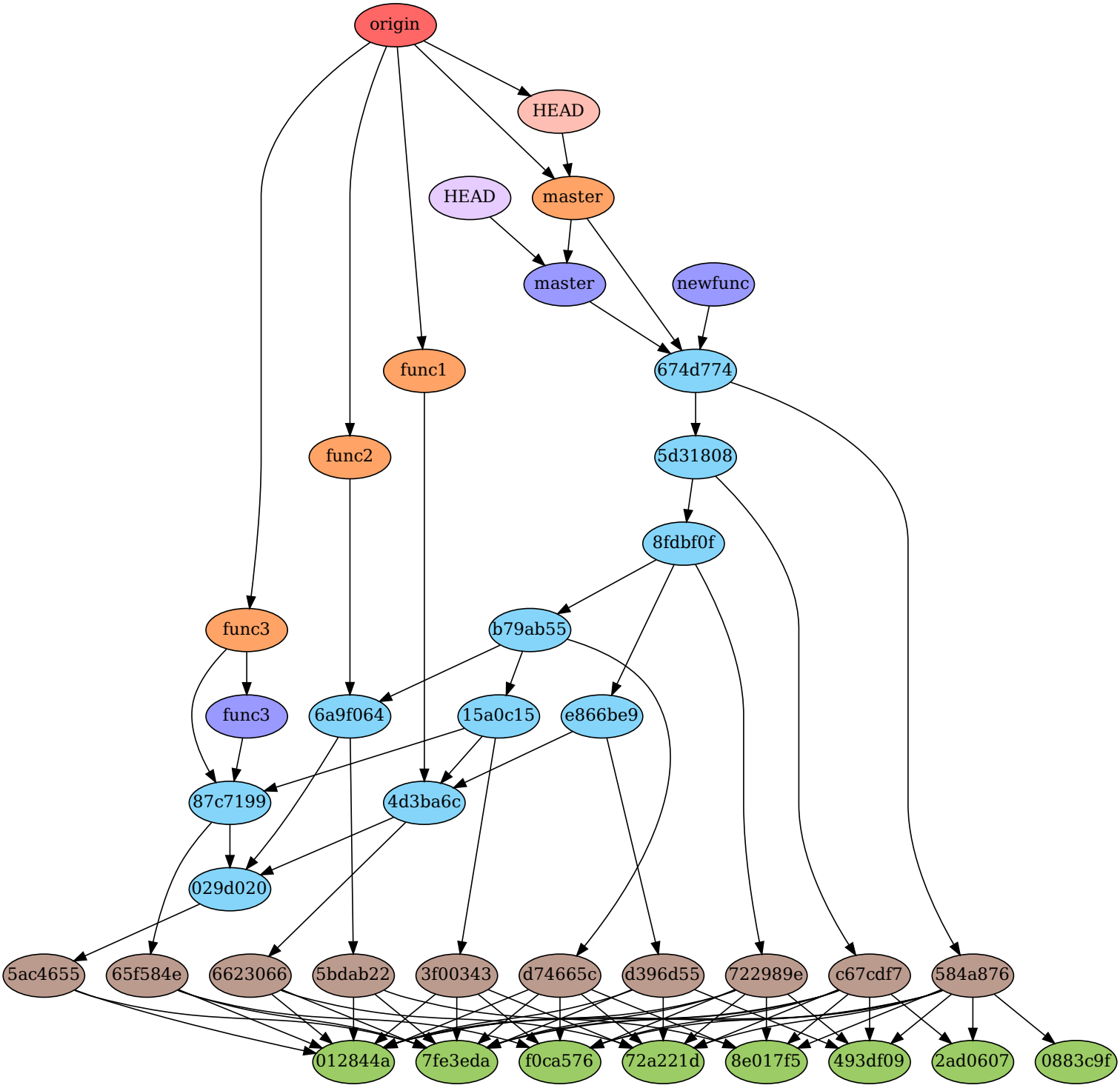

HEAD
master
f189096
feature
2e793ac
42c04ea
40573d0
6df49dd
3b58409
8e41736
ef06fb2
14be0d4
e69de29
ef55c6c
89e0427
e51706c
2e81171
15b8f2a
45b983b
ce01362


HEAD
master
b5a349e
27271a0
1ff761d
feature
40573d0
6df49dd
1df6103
c5c3e32
14be0d4
89e0427
2c91566
3b58409
8105146
15b8f2a
e69de29
636ffab
e51706c
2e81171
45b983b
ce01362

4(i),(ii). The git graphs are in the following order:
(a) Final git graph
(b) Git graph after rebasing 4f
(c) Git graph after rebasing 4g


origin
HEAD
master
master
func2
func3
c2406a1
674d774
5d31808
8fdbf0f
func1
b79ab55
6a9f064
func1
15a0c15
e866be9
87c7199
4d3ba6c
029d020
5ac4655
65f584e
6623066
5bdab22
3f00343
d396d55
d74665c
722989e
c67cdf7
584a876
a65231a
012844a
7fe3eda
72a221d
f0ca576
8e017f5
493df09
2ad0607
0883c9f
e691af1


origin
HEAD
func3
HEAD
master
bugFix
master
5d31808
func1
8fdbf0f
func2
b79ab55
func2
e866bc9
15a0c15
6a9f064
4d3ba6c
87c7199
029d020
c67cdf7
722989e
d396d55
d74665c
3f00343
6623066
5bdab22
65f584e
5ac4655
2ad0607
493df09
8e017f5
72a221d
f0ca576
012844a
7fe3eda


origin
HEAD
HEAD
master
master
newfunc
func1
674d774
func2
5d31808
8fdbf0f
func3
b79ab55
func3
6a9f064
15a0c15
e866be9
87c7199
4d3ba6c
029d020
5ac4655
65f584e
6623066
5bdab22
3f00343
d74665c
d396d55
722989e
c67cdf7
584a876
012844a
7fe3eda
f0ca576
72a221d
8e017f5
493df09
2ad0607
0883c9f

4(iii). Yes it can be done simply by adding one more case in the switch. As per the code of Member1, there is loop which helps in printing the zig zag traversal therefore in a new case we can add another loop which prints the reverse zigzag as done by the group member 2.