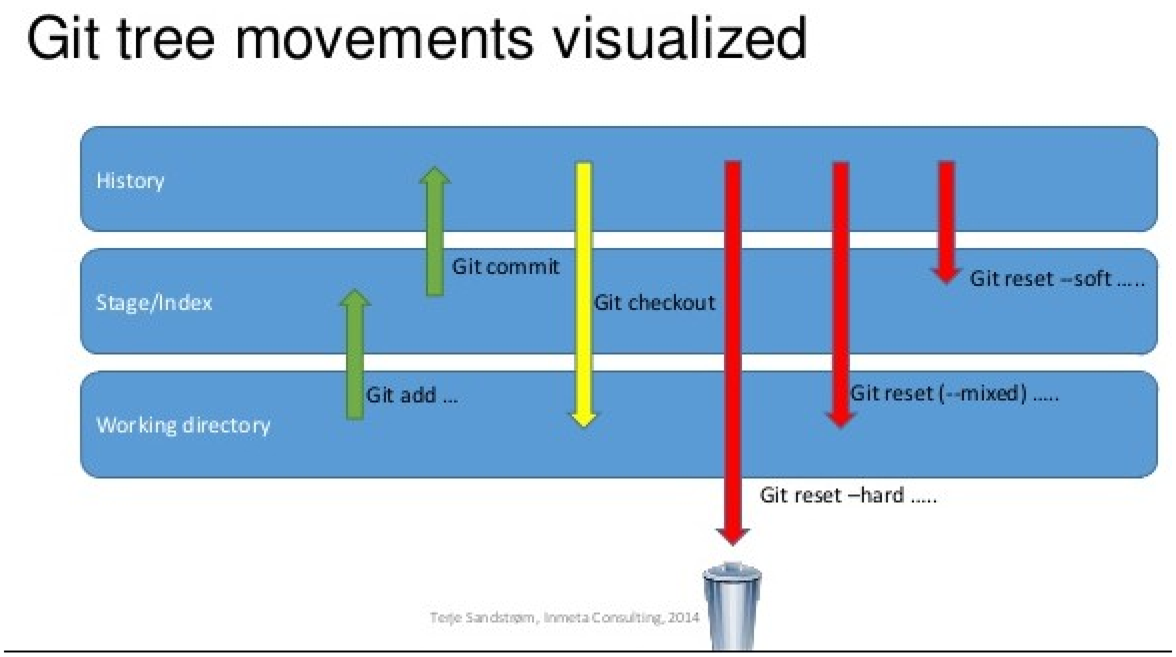
Git reset and git rebase:

Reset will bring down the changes to the commit specified.
There are three modes of resetting the changes : soft, mixed, hard



Now you should be able to understand the sequence of the below commands.

```
fms@git_practice:~/dev_flowers/sample-project$ vi new_file.txt
fms@git_practice:~/dev_flowers/sample-project$ cat new_file.txt
new file
fms@git_practice:~/dev_flowers/sample-project$

fms@git_practice:~/dev_flowers/sample-project$ git status
On branch branch_one
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    new_file.txt

nothing added to commit but untracked files present (use "git add" to track)
fms@git_practice:~/dev_flowers/sample-project$
```

```
fms@git_practice:~/dev_flowers/sample-project$ git log --oneline
b52fa47 (HEAD -> branch_one, origin/branch_one) new file candy.sh
3d1b34e (origin/master, origin/HEAD) Revert "created colors.java"
803fdf4 renamed flowers.c to buds.c
bcfcc4e remove fruits.c from this project
32960a2 update .gitignore to ignore html files
7d8d1d6 created sweets.py
66c27af created colors.java
544e581 updated fruits.c
a135ee5 created flowers.cpp
36c5a24 created a file fruits.c
fms@git_practice:~/dev_flowers/sample-project$
```

We are resetting to 4th commit.

```
fms@git_practice:~/dev_flowers/sample-project$ git reset --soft bcfcc4e
fms@git_practice:~/dev_flowers/sample-project$
```

```
fms@git_practice:~/dev_flowers/sample-project$ git status
On branch branch_one
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    flowers.cpp -> buds.cpp
    new file:   candy.sh
    deleted:    colors.java

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

    new_file.txt
```

The commits made on top of this commit id are brought back to staging area.

```
fms@git_practice:~/dev_flowers/sample-project$ cat .git/COMMIT_EDITMSG
new file candy.sh
```

We see that HEAD is still pointing to **b52fa47** and not **bcfcc4e** after doing soft reset.

Let us check mixed mode.

```
fms@git_practice:~/dev_flowers/sample-project$ git reset --mixed
bcfcc4e
Unstaged changes after reset:
D    colors.java
D    flowers.cpp
fms@git_practice:~/dev_flowers/sample-project$

fms@git_practice:~/dev_flowers/sample-project$ git status
On branch branch_one
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    deleted:    colors.java
    deleted:    flowers.cpp

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

    buds.cpp
    candy.sh
    new_file.txt

no changes added to commit (use "git add" and/or "git commit -a")
fms@git_practice:~/dev_flowers/sample-project$
```

The files in the commits are brought back to working area. Hence they are listed under untracked.

```
fms@git_practice:~/dev_flowers/sample-project$ git reset --hard bcfcc4e
HEAD is now at bcfcc4e remove fruits.c from this project
fms@git_practice:~/dev_flowers/sample-project$
```

This will reset all the files after this commit id. All the commits will be lost. This is usually used to clean up our workspace when we make an unnecessary commit that creates a mess.

```
fms@git_practice:~/dev_flowers/sample-project$ git status
On branch branch_one
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    buds.cpp
    candy.sh
    new_file.txt

nothing added to commit but untracked files present (use "git add" to track)
fms@git_practice:~/dev_flowers/sample-project$

fms@git_practice:~/dev_flowers/sample-project$ git log --oneline
bcfcc4e (HEAD -> branch_one) remove fruits.c from this project
32960a2 update .gitignore to ignore html files
7d8d1d6 created sweets.py
66c27af created colors.java
544e581 updated fruits.c
a135ee5 created flowers.cpp
36c5a24 created a file fruits.c
fms@git_practice:~/dev_flowers/sample-project$

fms@git_practice:~/dev_flowers/sample-project$ git branch
* branch_one
  master
fms@git_practice:~/dev_flowers/sample-project$
```

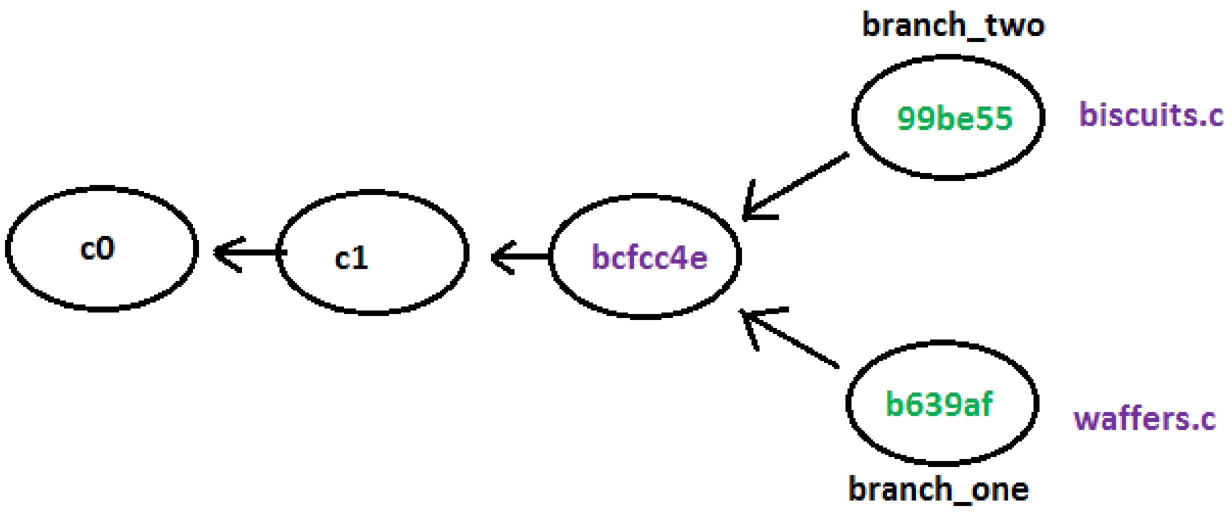
We are on branch_one and our latest commit is bcfcc4e
From this commit bcfcc4e, we will checkout to a new branch branch_two

```
fms@git_practice:~/dev_flowers/sample-project$ git checkout -b
branch_two
Switched to a new branch 'branch_two'
fms@git_practice:~/dev_flowers/sample-project$

fms@git_practice:~/dev_flowers/sample-project$ git branch
  branch_one
* branch_two
  master
fms@git_practice:~/dev_flowers/sample-project$
fms@git_practice:~/dev_flowers/sample-project$ vi biscuits.txt
fms@git_practice:~/dev_flowers/sample-project$ cat biscuits.txt
biscuits
fms@git_practice:~/dev_flowers/sample-project$
```

```
fms@git_practice:~/dev_flowers/sample-project$ git add biscuits.txt
fms@git_practice:~/dev_flowers/sample-project$ git commit -m
"biscuits.txt on branch_two"
[branch_two c6d0e60] biscuits.txt on branch_two
 1 file changed, 1 insertion(+)
 create mode 100644 biscuits.txt
fms@git_practice:~/dev_flowers/sample-project$

fms@git_practice:~/dev_flowers/sample-project$ git log -2
commit 99be55cfb582370bb6cc9e38d4cda55e95c4a7a1 (HEAD ->
branch_two)
Author: Dev-Flowers <dev_flowers@FLOWERS.com>
Date:   Tue Nov 6 16:58:21 2018 +0530

    biscuits.txt on branch_two

commit bcfcc4e724eb20ef137158ffbddba3e76ce5879f
Author: Dev-Colors <dev_colors@COLORS.com>
Date:   Tue Nov 6 15:14:31 2018 +0530

    remove fruits.c from this project
fms@git_practice:~/dev_flowers/sample-project$

fms@git_practice:~/dev_flowers/sample-project$ git checkout
branch_one
Switched to branch 'branch_one'
fms@git_practice:~/dev_flowers/sample-project$

fms@git_practice:~/dev_flowers/sample-project$ vi waffers.txt
fms@git_practice:~/dev_flowers/sample-project$ cat waffers.txt
waffers
fms@git_practice:~/dev_flowers/sample-project$

fms@git_practice:~/dev_flowers/sample-project$ git add waffers.txt
fms@git_practice:~/dev_flowers/sample-project$ git commit -m
"waffers.txt on branch_one"
[branch_one b639afa] waffers.txt on branch_one
 1 file changed, 1 insertion(+)
 create mode 100644 waffers.txt
fms@git_practice:~/dev_flowers/sample-project$
```

```
fms@git_practice:~/dev_flowers/sample-project$ git log -2
commit b639afa2d85a76f16325455cc7093b8a50d5da56 (HEAD ->
branch_one)
Author: Dev-Flowers <dev_flowers@FLOWERS.com>
Date:   Tue Nov 6 17:15:41 2018 +0530

    waffers.txt on branch_one

commit bcfcc4e724eb20ef137158ffbddba3e76ce5879f
Author: Dev-Colors <dev_colors@COLORS.com>
Date:   Tue Nov 6 15:14:31 2018 +0530

    remove fruits.c from this project
fms@git_practice:~/dev_flowers/sample-project$
fms@git_practice:~/dev_flowers/sample-project$ git checkout
branch_one
Switched to branch 'branch_one'
fms@git_practice:~/dev_flowers/sample-project$
```

So on top of commit bcfcc4e, we are creating one more commit on both the branches.
On branch branch_two -> biscuits.c is committed
On branch branch_one -> waffers.c is committed



We are now on branch_one and trying to rebase the changes made on branch_two. In the sense we are trying to merge biscuits.c commit on wafers.c commit.

```
fms@git_practice:~/dev_flowers/sample-project$ git rebase branch_two
First, rewinding head to replay your work on top of it...
Applying: waffers.txt on branch_one
fms@git_practice:~/dev_flowers/sample-project$

fms@git_practice:~/dev_flowers/sample-project$ git log -3
commit b7ef981865d35b30ee1f94bb9b24880522992e1a (HEAD ->
branch_one)
Author: Dev-Flowers <dev_flowers@FLOWERS.com>
Date:   Tue Nov 6 17:15:41 2018 +0530

    waffers.txt on branch_one

commit 99be55cfb582370bb6cc9e38d4cda55e95c4a7a1 (branch_two)
Author: Dev-Flowers <dev_flowers@FLOWERS.com>
Date:   Tue Nov 6 16:58:21 2018 +0530

    biscuits.txt on branch_two

commit bcfcc4e724eb20ef137158ffbddba3e76ce5879f
Author: Dev-Colors <dev_colors@COLORS.com>
Date:   Tue Nov 6 15:14:31 2018 +0530

    remove fruits.c from this project
fms@git_practice:~/dev_flowers/sample-project$
```

Done. This is how rebase works. The base commit may vary. At times rebase will throw error when there is a conflict.

We will do a hard reset and make our HEAD point to the commit that we did before rebase.

```
fms@git_practice:~/dev_flowers/sample-project$ git reset --hard
b7ef981865d35b30ee1f94bb9b24880522992e1a
HEAD is now at b7ef981 waffers.txt on branch_one
fms@git_practice:~/dev_flowers/sample-project$

fms@git_practice:~/dev_flowers/sample-project$ vi waffers.txt
fms@git_practice:~/dev_flowers/sample-project$ git branch
* branch_one
  branch_two
  master
fms@git_practice:~/dev_flowers/sample-project$

fms@git_practice:~/dev_flowers/sample-project$ git add waffers.txt
fms@git_practice:~/dev_flowers/sample-project$ git commit -m
"new_waffers on branch_one"
[branch_one 14c9a13] new_waffers on branch_one
 1 file changed, 1 insertion(+), 1 deletion(-)
fms@git_practice:~/dev_flowers/sample-project$
```

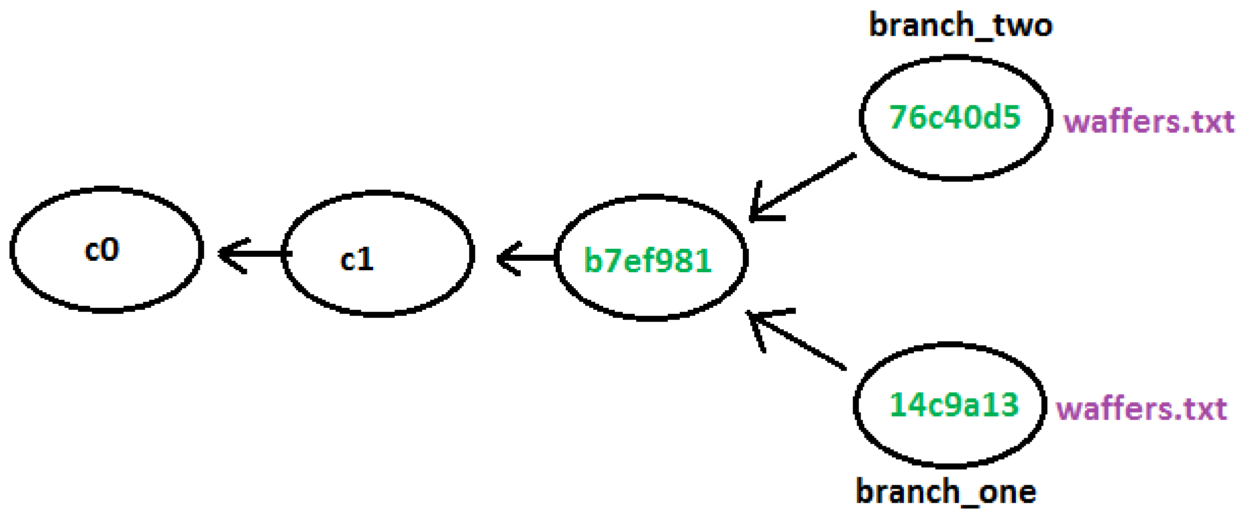
On branch_one, we are committing a new file ‘waffers.txt’.

```
fms@git_practice:~/dev_flowers/sample-project$ git checkout
branch_two
Switched to branch 'branch_two'
fms@git_practice:~/dev_flowers/sample-project$

fms@git_practice:~/dev_flowers/sample-project$ git log -2
commit 99be55cfb582370bb6cc9e38d4cda55e95c4a7a1 (HEAD ->
branch_two)
Author: Dev-Flowers <dev_flowers@FLOWERS.com>
Date:   Tue Nov 6 16:58:21 2018 +0530

    biscuits.txt on branch_two

commit bcfcc4e724eb20ef137158ffbddba3e76ce5879f
Author: Dev-Colors <dev_colors@COLORS.com>
Date:   Tue Nov 6 15:14:31 2018 +0530

    remove fruits.c from this project
fms@git_practice:~/dev_flowers/sample-project$
```

On branch_two, create a file with same name ‘waffers.txt’.

```
fms@git_practice:~/dev_flowers/sample-project$ vi waffers.txt
fms@git_practice:~/dev_flowers/sample-project$ git add waffers.txt
fms@git_practice:~/dev_flowers/sample-project$

fms@git_practice:~/dev_flowers/sample-project$ git commit -m
"waffers.txt is created again in branch_two"
[branch_two 76c40d5] waffers.txt is created again in branch_two
```

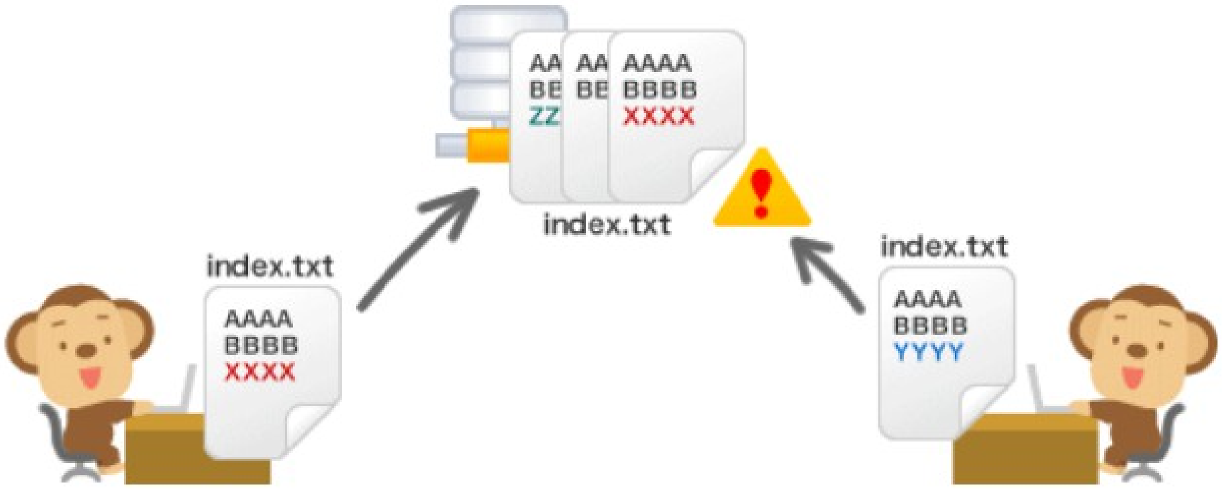
```
 1 file changed, 1 insertion(+)
 create mode 100644 waffers.txt
fms@git_practice:~/dev_flowers/sample-project$


fms@git_practice:~/dev_flowers/sample-project$ git log -2
commit 76c40d52d98b817ba2b0aeb1c50537b83214e33e (HEAD ->
branch_two)
Author: Dev-Flowers <dev_flowers@FLOWERS.com>
Date:   Tue Nov 6 17:24:32 2018 +0530

    waffers.txt is created again in branch_two

commit 99be55cfb582370bb6cc9e38d4cda55e95c4a7a1
Author: Dev-Flowers <dev_flowers@FLOWERS.com>
Date:   Tue Nov 6 16:58:21 2018 +0530

    biscuits.txt on branch_two
fms@git_practice:~/dev_flowers/sample-project$
```



From branch_two, we are trying to rebase it to branch_one.
Since we have modified the same file, rebase will result in merge conflict.

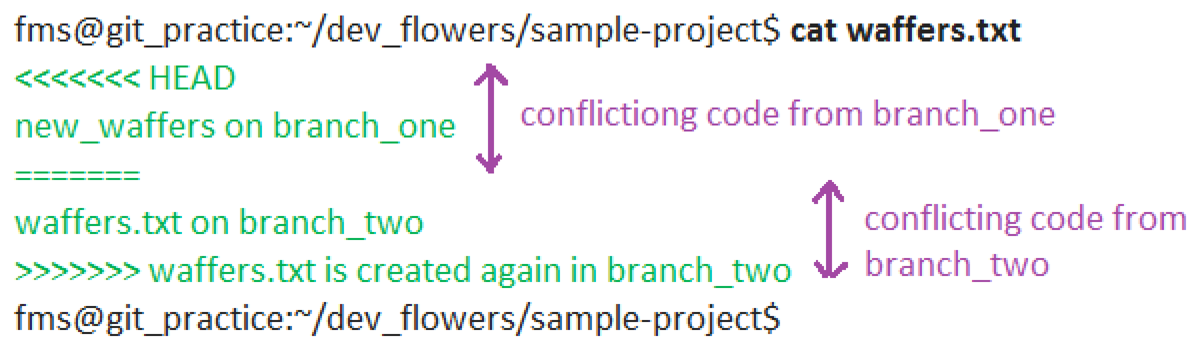
```
fms@git_practice:~/dev_flowers/sample-project$ git rebase branch_one
First, rewinding head to replay your work on top of it...
Applying: waffers.txt is created again in branch_two
Using index info to reconstruct a base tree...
```

```
Falling back to patching base and 3-way merge...
Auto-merging waffers.txt
CONFLICT (add/add): Merge conflict in waffers.txt
error: Failed to merge in the changes.
Patch failed at 0001 waffers.txt is created again in branch_two
Use 'git am --show-current-patch' to see the failed patch

Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase
--abort".

fms@git_practice:~/dev_flowers/sample-project$
```



```
fms@git_practice:~/dev_flowers/sample-project$ cat waffers.txt
<<<<<<< HEAD
new_waffers on branch_one
=======
waffers.txt on branch_two
>>>>>>> waffers.txt is created again in branch_two
fms@git_practice:~/dev_flowers/sample-project$
```

Git cannot auto resolve the conflict. So we have to choose which part of the code to retain and which part to skip. Edit the file, manually resolve the conflict and then commit the changes.



```
fms@git_practice:~/dev_flowers/sample-project$ cat waffers.txt
<<<<<<< HEAD
new_waffers on branch_one
=======
waffers.txt on branch_two
>>>>>>> waffers.txt is created again in branch_two
fms@git_practice:~/dev_flowers/sample-project$
```

conflictiong code from branch_one
conflicting code from branch_two

Git commands covered so far:

28. git reset
29. git rebase

Reference links for further study on git:

http://www.yolinux.com/TUTORIALS/Git-commands.html
https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf