ommand Summary (Undoing changes in a Git repository)

**LECTURE - Checking out commits in a Git repository - Part 1**

1. $ cd git-fast
2. # created a fresh git repository named "demo-checkout-commit" using initializr.zip
3. # Build up a commit history by modifying the file robots.txt, say 5 times and committing each time
4. # Let us assume we have one commit whose id/hash is 91770af
5. # the choice of commit id 91770af is arbitrary
7. # note where the HEAD is now (since HEAD will change later while doing a checkout)
8. $ git log --oneline
10. # clear the screen
11. # The command below will put you in a detached HEAD state.
12. # HEAD no longer points to a branch; it points directly to a commit.
13. $ git checkout 91770af

LECTURE - Checking out commits in a Git repository - Part 2

1. # you are in detached HEAD state and inside git repository (demo-checkout-commit)
2. # You can see only commits till commit-id 91770af
3. $ git log --oneline
5. # now edit and do a express commit for file robots.txt
6. # note down your new commit id/hash (Say ID1)
7. $ git log --oneline
9. $ git checkout master
11. # changes made in detached HEAD state is no more there
12. # the new commit ID1 is not visible
13. $ git log --oneline
15. # again you are in detached HEAD state
16. $ git checkout 91770af
18. # the new commit ID1 is not visible which means earlier changes...
19. # in detached HEAD state were not preserved
20. $ git log --oneline
22. # now we want to retain our commit i.e, preserve our changes
23. # make changes in the robots.txt and do an express commit
25. # create a branch named weird-experiment
26. $ git branch weird-experiment
28. $ git checkout master
29. # the new commit id is not visible here
30. # you can check the file robots.txt using cat command also
31. $ git log --oneline
33. # new commit id is visible now
34. # you can examine the file robots.txt using cat command
35. $ git checkout weird-experiment

**LECTURE - Checking out files**

1. # cd to git repository (demo-checkout-commit) used in previous lecture
2. $ git log --online
4. # Let us assume we have the commit-id 91770af
6. # step-2: examine the current contents of the file
7. $ cat robots.txt
9. # now we will able to see the content of the file as at commit-id 91770af
10. $ git checkout 91770af robots.txt
12. # this will show the file has been added to staging area
13. $ git status
15. # step-5: examine the contents of the file (should be different from step-2)
16. $ cat robots.txt
18. # if we do not want to revert back the file robots.txt
19. $ git checkout HEAD robots.txt
21. # you will see a clean working directory
22. $ git status
24. # now we want to revert back to previous version of robots.txt as at commit-id 91770af
25. $ git checkout 91770af robots.txt
26. $ git commit -am "reverting back file state to commit 91770af"
28. # clean working directory
29. $ git status

32. # file is reverted and the contents are as seen in step-5
33. $ cat robots.txt

**LECTURE - Reverting changes**

1. # cd into repository "undo-demo-git-repo"
3. # add the text say, "line 5"
4. $ vim checkoutfile.txt
6. # modified stage visible
7. $ git status
9. # commit the above change
10. $ git commit -am checkoutfile.txt "commit message..."
12. $ git log --oneline
13. $ cat checkoutfile.txt
15. # Now let's assume the earlier commit is a buggy one and so we will revert it
16. # revert command introduces a new commit in a safe manner (opens the default editor)
17. $ git revert HEAD
19. # check commit history - a new commit for revert
20. $ git log --oneline
22. # note the change made earlier has been reverted "line 5" is not visible
23. $ cat checkoutfile.txt

**LECTURE - Resetting Git repository - Part1**

1. # we are in repository git-reset-demo-1 with 2 files goldies-time-table and jimmys-time-table
2. # the repository has an existing commit history
4. # now edit file jimmys-time-table and add a line at the end
5. $ vim jimmys-time-table
7. # add the change to staging area
8. $ git add jimmys-time-table
10. # change in staging area is visible
11. $ git status
13. # undoes changes in staging area
14. $ git reset jimmys-time-table
16. # changes in staging area is removed; file in modified state
17. $ git status
19. # the changed in working directory is intact
20. $ cat jimmys-time-table

**LECTURE - Resetting Git repository - Part2**

1. # we are in repository git-reset-demo-2 (this repo is clone of git-reset-demo-1)
3. # edit file and add one line at the end
4. $ vim jimmys-time-table
6. # edit file and add one line at the end
7. $ vim goldies-time-table
9. # files are in modified state
10. $ git status
12. # add both files to staging area
13. $ git add .
15. # changes visible in staging area
16. $ git status
18. # "reset" command removes changes in staging area leaving the working directory intact
19. $ git reset
21. # both files in modified state
22. $ git status
24. #######################################
25. # git reset demo using the --hard option
26. # we are still in git-reset-demo-2 repository(this repo is clone of git-reset-demo-1)
28. # files are in modified state
29. $ git status
31. # add both files to staging area
32. $ git add .
34. # changes visible in staging area
35. $ git status
37. # --hard option removes all changes in working directory and staging area
38. $ git reset --hard
40. # the working directory is clean since all changes have been reset in working dir and staging
41. $ git status

**LECTURE - Resetting Git repository - Part3**

1. # we are inside git-reset-demo-4 repository (this repo is clone of git-reset-demo-1)
3. # Let's say, we have 5 commits in the commit history
4. # let' say, the 5th commit-id (HEAD) is 9166e4f and 4th commit-id is d21a539
5. $ git log --oneline
7. # let's reset the repository to 4th commit-id
8. $ git reset d21a539
10. # the 5th commit-id(9166e4f) has been removed from history and the HEAD is now at d21a539
11. $ git log --oneline
13. # the files are in modified state and changes in staging area has been removed
14. # the files are jimmys-time-table and goldies-time-table
15. $ git status
17. # now we will commit the above changes in working dir in 2 smaller chunks
18. $ git add jimmys-time-table
19. $ git commit -m "commit message....."
21. # now only goldies-time-table changes are visible in modified state
22. $ git status
24. $ git add goldies-time-table
25. $ git commit -m "commit message....."
27. # clean working directory
28. $ git status
30. $ git log --oneline
31. # we are in git-reset-demo-5 repository (this repo is clone of git-reset-demo-1)
33. # let's say there are 5 commits
34. $ git log --oneline
36. # now we will reset to the 3rd commit-id a53f51c (2 commits before HEAD) with the --hard option
37. # this command will remove 4th & 5th commit-ids as well removed all changes in working dir and staging area
38. $ git reset --hard a53f51c
40. # the 4th and 5th commit-ids has been removed from history
41. $ git log --oneline
43. # examine the file contents to confirm things
44. # the lines pertaining to 4th and 5th commit-ids has been removed
45. $ cat goldies-time-table
46. $ cat jimmys-time-table

**LECTURE - Cleaning Git repository**

1. $ cd git-fast
2. $ mkdir git-clean-demo
3. $ cd git-clean-demo
5. # create an empty repository
6. $ git init
8. # create few empty files
9. $ touch clean-demo-file
10. $ touch clean-demo-tracked-file
12. $ git add clean-demo-tracked-file
13. $ git commit -m "commit tracked file"
15. # shows clean-demo-file as an untracked file
16. $ git status
18. # -n option allows to make a dry run of "git clean" command
19. $ git clean -n
21. #################################################
23. # -f option (-f means force) removes untracked files from current directory
24. # so this will remove clean-demo-file
25. $ git clean -f
27. #################################################
29. $ mkdir clean-demo-dir
30. $ cd clean-demo-dir
32. # create an untracked file
33. $ touch clean-demofile-2
35. $ cd ..
37. # removes untracked file in path clean-demo-dir
38. $ git clean -f clean-demo-dir/
40. #confirm untracked file cleaning
41. $ cd clean-demo-dir
43. # "ls" produces no results meaning the untracked file was cleaned
44. $ ls
46. #################################################
47. $ cd ..
48. $ ls
49. $ touch clean-demo-file-2
50. $ ls
52. # -df option cleans both untracked files and directories
53. $ git clean -df
54. $ ls
56. #################################################
57. # create .gitignore file and add the file name "clean-demo-file-5"
58. $ vim .gitignore
59. $ git add .gitignore
60. $ git commit -m "committing .gitignore file"
62. $ touch clean-demo-file-4
63. $ touch clean-demo-file-5
64. $ ls -al
66. # -xf option cleans all untracked files including those mentioned in .gitignore file
67. $ git clean -xf
69. $ ls