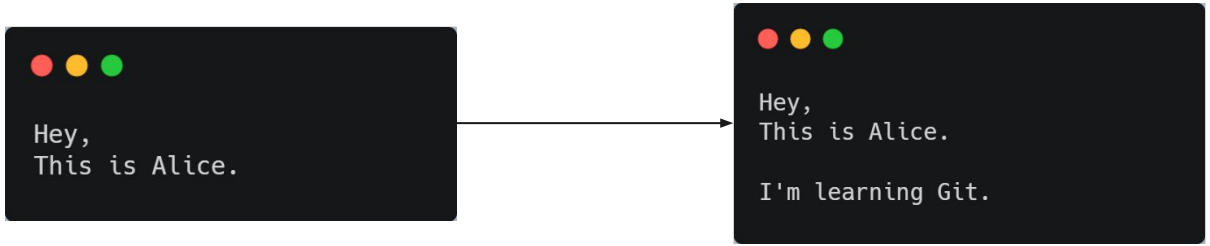
Git

Undoing Things

Undoing Changes before committing

`git checkout <filename>` is used to revert to previous changes that is not staged.




```
ssingh_raviprakash@cloudshell:~/tutorial$ git checkout Alice.txt
ssingh_raviprakash@cloudshell:~/tutorial$
```

If you want to checkout individual changes instead of whole file use -p flag.

Undoing Things

Viewing an Old Version

You can use git checkout to visit any old commit.

```
b7119f2 Continue doing crazy things
872fa7e Try something crazy
a1e8fb5 Make some important changes to hello.txt
435b61d Create hello.txt
9773e52 Initial import
```

```
git checkout a1e8fb5
```

```
git checkout main
```

Undoing Things

Undoing Changes before committing

git reset is used to revert to previous changes that are staged.

git reset HEAD <filename> removes specific file from staging area.

git reset . removes all file from staging area

```
ssingh_raviprakash@cloudshell:~/tutorial$ touch temp.txt
ssingh_raviprakash@cloudshell:~/tutorial$ git add *
ssingh_raviprakash@cloudshell:~/tutorial$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   temp.txt

ssingh_raviprakash@cloudshell:~/tutorial$ git reset HEAD temp.txt
```

Undoing Things

Undoing Changes before committing

```
git reset --hard
```

Resets staging area and working directory to most recent commit

Git reset command can be used in three ways:

- Hard
- Soft
- Mixed

Undoing Things

Undoing Changes before committing

Hard : `git reset --hard HEAD~1`

Soft : `git reset --soft HEAD~1`

Mixed : `git reset --mixed HEAD~1`

Undoing Things

Amending Commits

git commit --amend command is used to modify the most recent commit

```
git commit --amend
```

- It lets you combine staged changes with the previous commit instead of creating an entirely new commit.
- It can also be used to simply edit the previous commit message without changing its snapshot.

Undoing Things

Amending Commits

Suppose we forgot to add both files in commit

```
ssingh_raviprakash@cloudshell:~/tutorial$ touch Session1.txt
ssingh_raviprakash@cloudshell:~/tutorial$ touch Session2.txt
ssingh_raviprakash@cloudshell:~/tutorial$ git add Session1.txt
ssingh_raviprakash@cloudshell:~/tutorial$ git commit -m "Added 2 files"
[master c9105f3] Added 2 files
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Session1.txt
ssingh_raviprakash@cloudshell:~/tutorial$
```

Simply add 2nd file to Staging area and amend commit

```
ssingh_raviprakash@cloudshell:~/tutorial$ git add Session2.txt
ssingh_raviprakash@cloudshell:~/tutorial$ git commit --amend
```

Undoing Things

Amending Commits

You will be prompted with your preferred editor with commit message and files to be committed.

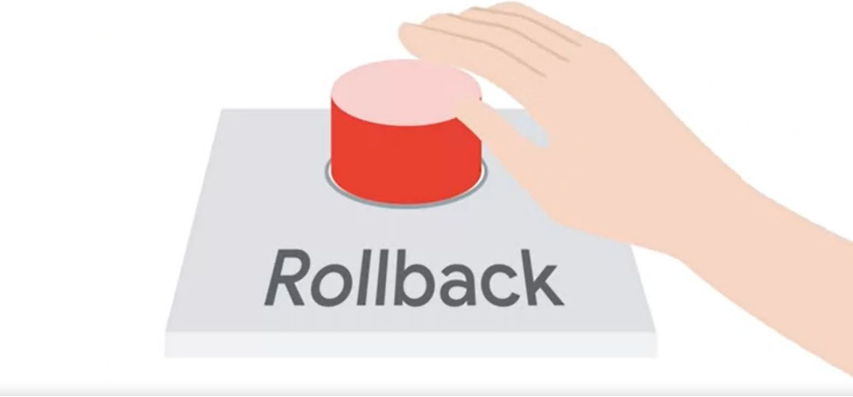
```
  GNU nano 3.2                    /home/ssingh_raviprakash/tutorial/.git/COMMIT_EDITMSG

Added 2 files

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Sat Jun 12 12:26:13 2021 +0000
#
# On branch master
# Changes to be committed:
#       new file:   Session1.txt
                                                        [ Read 12 lines ]
^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     M-U Undo       M-A Mark Text  M-] To Bracket  M-Q Previous
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line  M-E Redo       M-6 Copy Text  ^Q Where Was    M-W Next
```

Change Commit message accordingly and save the file.

It is generally discouraged to use amend in Public Repos. cause it changes Git history.

Undoing Things

Rollbacks



It is generally discouraged to use amend in Public Repos. cause it changes Git history.

Undoing Things

Rollbacks

git revert command is used to revert commit.

- This creates a commit that contains the invert of all the changes made in bad commit.
- This prevents Git from losing history, which is important for the integrity of your revision history and for reliable collaboration.

Undoing Things

Rollbacks

Let's add a faulty commit and revert it.

```
Hey,
This is Alice.
```

```
Hey,
This is Alice.

This is Bob's file.
```

```
ssingh_raviprakash@cloudshell:~/tutorial$ git add Alice.txt
ssingh_raviprakash@cloudshell:~/tutorial$ git commit -m "Added a line"
[master 2358f7a] Added a line
 1 file changed, 3 insertions(+), 1 deletion(-)
ssingh_raviprakash@cloudshell:~/tutorial$
```

Undoing Things

Rollbacks

Now apply revert command to undo the commit.

```
ssingh_raviprakash@cloudshell:~/tutorial$ git revert HEAD
```

```
  GNU nano 3.2                    /home/ssingh_raviprakash/tutorial/.git/COMMIT_EDITMSG                    Modified

Revert "Added a line"

This reverts commit 2358f7a646c4e737921dd513e1a5fbe786cb3d25.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#       modified:   Alice.txt
#

^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     M-U Undo       M-A Mark Text  M-] To Bracket  M-Q Previous
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line  M-E Redo       M-6 Copy Text  ^Q Where Was    M-W Next
```

```
[master 36d601d] Revert "Added a line"
 1 file changed, 1 insertion(+), 3 deletions(-)
ssingh_raviprakash@cloudshell:~/tutorial$
```

Undoing Things

Identifying a Commit

What if we need to revert commit farther back in time?

We can target a specific commit using it's **commit id**

Undoing Things

Identifying a Commit

```
ssingh_raviprakash@cloudshell:~/tutorial$ git revert d9ae
```

```
  GNU nano 3.2          /home/ssingh_raviprakash/tutorial/.git/COMMIT_EDITMSG
  GNU nano 3.2                                  /home/ssingh_raviprakash/tutorial/.git/COMMIT_EDITMSG                    Modified

Revert "Added 2 files"

We no longer need those files

This reverts commit d90ae9f4148d1bf8dc9f1d9b952380b8455fb453.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#       deleted:    Session1.txt

^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     M-U Undo       M-A Mark Text  M-] To Bracket M-Q Previous
```

```
commit 2358f7a646c4e737921dd513e1a5fbe786cb3d25
Author: Ravi Prakash Singh <ssingh.raviprakash@gmail.com>
Date:   Mon Jun 14 03:42:20 2021 +0000
[master 5788780] Revert "Added 2 files"
 2 files changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 Session1.txt
 delete mode 100644 Session2.txt
ssingh_raviprakash@cloudshell:~/tutorial$
```

Undoing Things

Cheat Sheet

- **git checkout** is effectively used to switch branches.
- **git reset** basically resets the repo, throwing away some changes.

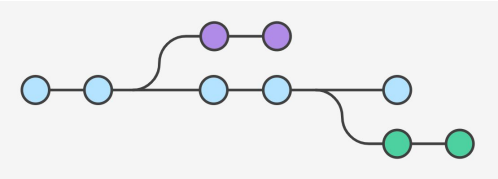
There are some other useful articles online, which discuss more aggressive approaches to resetting the repo.

- **git commit --amend** is used to make changes to commits after-the-fact, which can be useful for making notes about a given commit.
- **git revert** makes a new commit which effectively rolls back a previous commit. It's a bit like an undo command

Branching and Merging

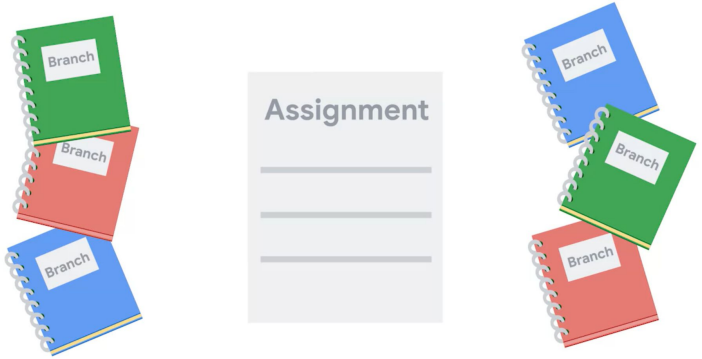
What is a Branch?

A branch is a pointer to a particular commit



It represents independent line of development in a project.

The default branch which git creates is **master.**



Branching and Merging

Creating a Branch

git branch command is used to create, delete and manipulate branches.

```
git branch
```

git branch is used to list all branches

```
git branch <branch>
```

Create a new branch

```
git branch -m <branch>
```

Rename the current branch to <branch>

Branching and Merging

Working with Branches

git checkout command is used to switch between branches.

```
ssingh_raviprakash@cloudshell:~/tutorial$ git branch classroom
ssingh_raviprakash@cloudshell:~/tutorial$ git checkout classroom
Switched to branch 'classroom'
ssingh_raviprakash@cloudshell:~/tutorial$
```

You can do this in single command using -b flag

```
git checkout -b <new-branch>
```

```
ssingh_raviprakash@cloudshell:~/tutorial$ git checkout -b StudyNotes
Switched to a new branch 'StudyNotes'
```

Branching and Merging

Switching Branch

git checkout command is used to switch between branches.

```
ssingh_raviprakash@cloudshell:~/tutorial$ git branch classroom
ssingh_raviprakash@cloudshell:~/tutorial$ git checkout classroom
Switched to branch 'classroom'
ssingh_raviprakash@cloudshell:~/tutorial$
```

You can do this in single command using -b flag

```
git checkout -b <new-branch>
```

```
ssingh_raviprakash@cloudshell:~/tutorial$ git checkout -b StudyNotes
Switched to a new branch 'StudyNotes'
```

Branching and Merging

Deleting Branch

To delete a branch use -d flag followed by branch name

```
git branch -D <branch>
```

```
ssingh_raviprakash@cloudshell:~/tutorial$ git status
On branch StudyNotes
nothing to commit, working tree clean
ssingh_raviprakash@cloudshell:~/tutorial$ git checkout master
Switched to branch 'master'
ssingh_raviprakash@cloudshell:~/tutorial$ git branch -d StudyNotes
Deleted branch StudyNotes (was 5788780).
ssingh_raviprakash@cloudshell:~/tutorial$
```

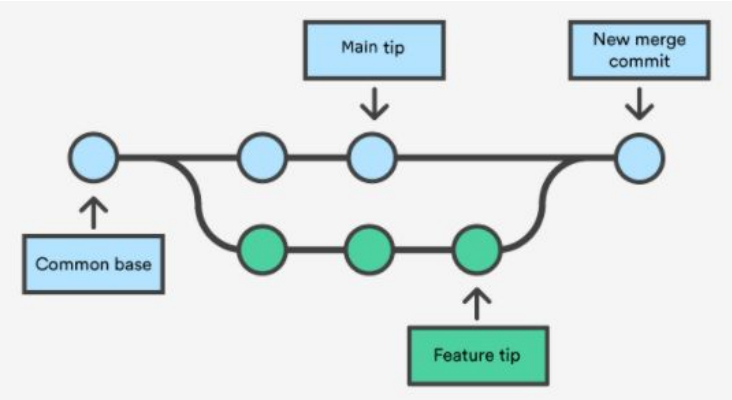
- If you try to delete a branch without merging into master git will raise warning

```
ssingh_raviprakash@cloudshell:~/tutorial$ git branch -d main
error: The branch 'main' is not fully merged.
If you are sure you want to delete it, run 'git branch -D main'.
ssingh_raviprakash@cloudshell:~/tutorial$
```

Branching and Merging

Merging

The term git uses for combining branched data and history together.

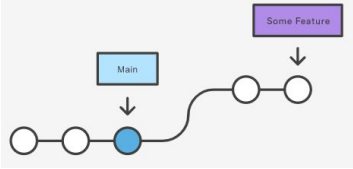


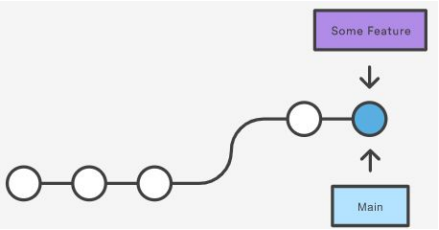
Branching and Merging

Merging

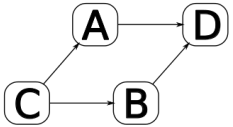
Git uses two different algorithms to perform a merge

- <u>Fast-Forward Merge</u>:- A fast-forward merge can occur when there is a linear path from the current branch tip to the target branch.





- <u>Three-Way Merge</u>:- A three-way merge is performed after an automated difference analysis between a file "A" and a file "B" while also considering the origin, or common ancestor, of both files "C".



Branching and Merging

Fast-Forward Merge

Git all has to do is fast-forward the current branch pointer to target branch.

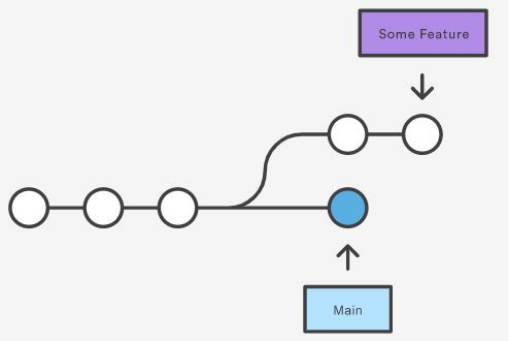
```
# Start a new feature
git checkout -b new-feature main
# Edit some files
git add <file>
git commit -m "Start a feature"
# Edit some files
git add <file>
git commit -m "Finish a feature"
# Merge in the new-feature branch
git checkout main
git merge new-feature
git branch -d new-feature
```

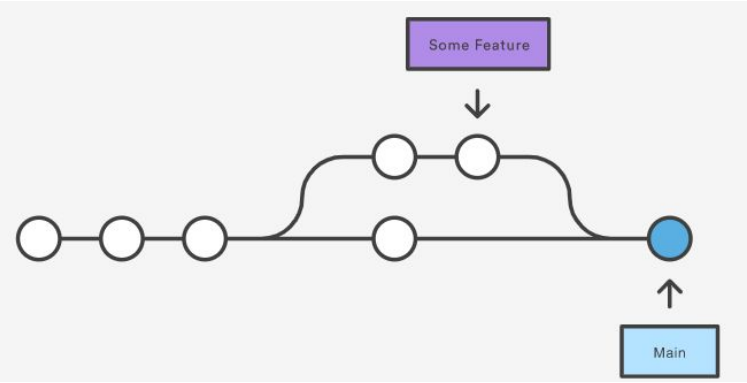
```
ssingh_raviprakash@cloudshell:~/tutorial$ git merge main
Updating 5788780..1adcaf6
Fast-forward
 main.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 main.txt
ssingh_raviprakash@cloudshell:~/tutorial$
```

Branching and Merging

Three-Way Merge

But what if the branches have diverged.





- 3-way merges use a dedicated commit to tie together the two histories.
- The nomenclature comes from the fact that Git uses three commits to generate the merge commit: the two branch tips and their common ancestor.

Branching and Merging

Three-Way Merge

Consider the following scenario, which requires three-way merge.

This is a common scenario for large features or when several developers are working on a project simultaneously.

```
Start a new feature
git checkout -b new-feature main
# Edit some files
git add <file>
git commit -m "Start a feature"
# Edit some files
git add <file>
git commit -m "Finish a feature"
# Develop the main branch
git checkout main
# Edit some files
git add <file>
git commit -m "Make some super-stable changes to main"
# Merge in the new-feature branch
git merge new-feature
git branch -d new-feature
```

Branching and Merging

How conflicts are presented?

```
here is some content not affected by the conflict
<<<<<<< main
this is conflicted text from main
=======
this is conflicted text from feature branch
>>>>>>> feature branch;
```

Alice.txt ×

```
1  Hey,
2  This is Alice.
3
   Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
4  <<<<<<< HEAD (Current Change)
5  This line was added in master branch.
6  =======
7  This line was added in Classroom branch.
8  >>>>>>> classroom (Incoming Change)
9
```

Branching and Merging

Cheat Sheet

Command	Explanation
git branch	Used to manage branches
git branch <name>	Creates the branch
git branch -d <name>	Deletes the branch
git branch -D <name>	Forcibly deletes the branch
git checkout <branch>	Switch to a branch
git checkout -b <branch>	Creates an new branch and switches to it
git merge <branch>	Merge joins branch together
git merge --abort	Used to abort merge action
git log --graph --oneline	Shows summarized view of commit history in repo

Key Takepoints

- Undoing Things
  - Undoing changes before committing
  - Amending Commits
  - Rollback
- Branching and Merging
  - What is Branch?
  - Creating Branches
  - Working with branches
  - Merging Conflicts