GIT Cheat sheet

Installation and Configuration

1. Download Git: https://git-scm.com/downloads
2. Choose your OS version and Install (For more: refer below Youtube links)
   a. For Windows: https://youtu.be/albr1o7Z1nw
   b. For Mac: https://youtu.be/sJ4zr0a4GAs
   c. For Linux: https://youtu.be/folbGtWNVwI

3. Basic Configuration to be done for the first time:
   a. Configure an User name
      git config --global user.name "Shivam Som"
   b. Configure an Email Id
      git config --global user.email "shivamsom3@gmail.com"

Basic Commands

1. Creating a Local Git Repository
   git init
   or
   Cloning a existing Git Repository
   git clone https://<URL>

2. Adding your changes to a Git Repository
   git add . (* to add all modified or created files)
   *git add **filename*** (* to add a specific modified or created file)

3. Commit your changes to a Git Repository
   git commit -m "Some useful message string"

4. Using Git status command
   git status

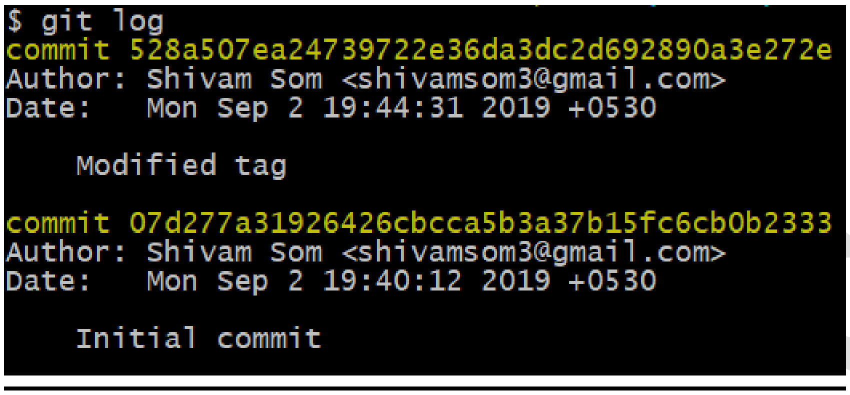
5. Checking the history
   git log

6. Configuring or adding a remote server url
   git remote add origin https://url

7. Pushing the changes to remote origin server
   git push -u origin <branch-name>

Reviewing changes:

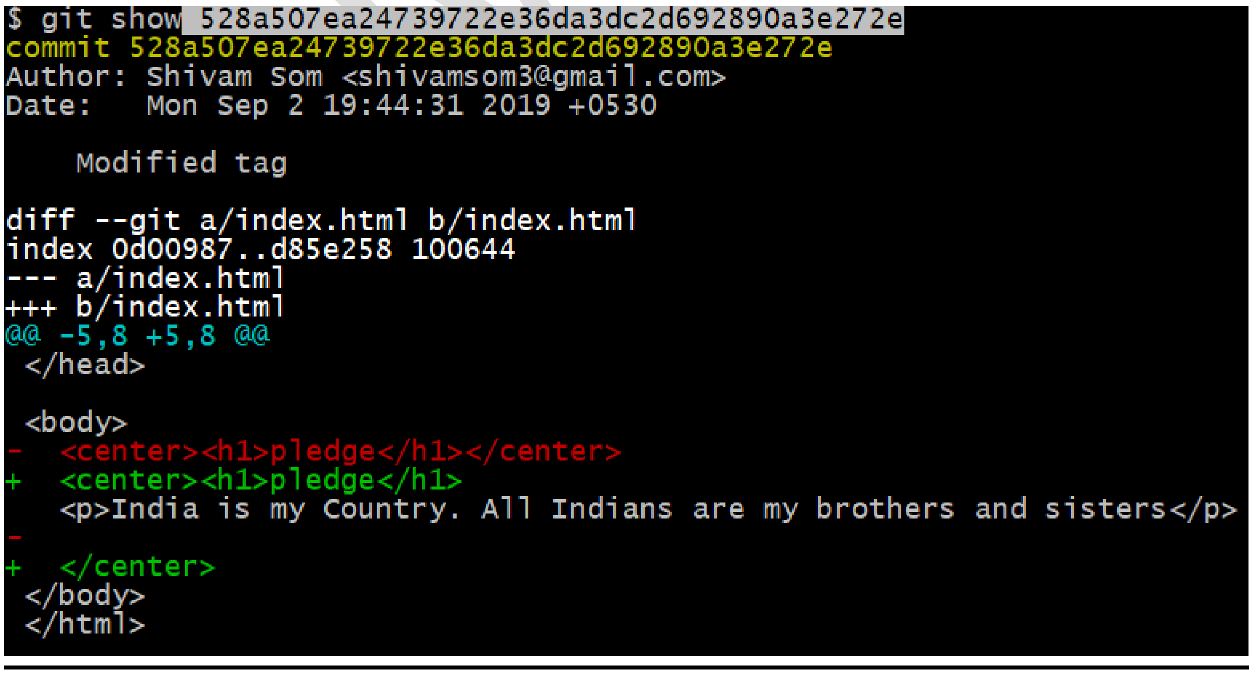
As we know to view the history of a file in git we use the command:
git log -> (Displays complete history of your repository, with Author, Date, Commit Id and Commit msg.)



```
$ git log
commit 528a507ea24739722e36da3dc2d692890a3e272e
Author: Shivam Som <shivamsom3@gmail.com>
Date:   Mon Sep 2 19:44:31 2019 +0530

    Modified tag

commit 07d277a31926426cbcca5b3a37b15fc6cb0b2333
Author: Shivam Som <shivamsom3@gmail.com>
Date:   Mon Sep 2 19:40:12 2019 +0530

    Initial commit
```

Sometimes, there might be a need to check what files are actually changed in particular commit

To do that,
git show ***commit-Id***
commit-id: It is a unique key or identifier that represents a particular commit in your repository.
Eg. git show 528a507ea24739722e36da3dc2d692890a3e272e



```
$ git show 528a507ea24739722e36da3dc2d692890a3e272e
commit 528a507ea24739722e36da3dc2d692890a3e272e
Author: Shivam Som <shivamsom3@gmail.com>
Date:   Mon Sep 2 19:44:31 2019 +0530

    Modified tag

diff --git a/index.html b/index.html
index 0d00987..d85e258 100644
--- a/index.html
+++ b/index.html
@@ -5,8 +5,8 @@
 </head>

 <body>
-   <center><h1>pledge</h1></center>
+   <center><h1>pledge</h1>
    <p>India is my Country. All Indians are my brothers and sisters</p>
-
+   </center>
 </body>
 </html>
```

Comparing Changes:

To compare the difference between your current changes and the changes that you made previously,

git diff

```
$ git diff

diff --git a/index.html b/index.html
index d85e258..c905c97 100644
--- a/index.html
+++ b/index.html
@@ -6,7 +6,9 @@

 <body>
   <center><h1>pledge</h1>
-  <p>India is my Country. All Indians are my brothers and sisters</p>
+  <p>India is my Country.</p>
+  <br>
+  <p> All Indians are my brothers and sisters</p>
   </center>
 </body>
 </html>
warning: LF will be replaced by CRLF in index.html.
The file will have its original line endings in your working directory.
```

Page Break

Working with Pull and Merge

Real time example on Pull and Merge

1. Two Developers working on project. Let's say John and Steven.
2. John adds following lines to a file called **styles.css**

```
body {
background-color: blue;
}
h1{
color: red;
}
```

3. John performs Add, Commit and then Push operations(ACP) as usual.
4. John then starts working on new business need.
5. By the time, John finishes his work on new business need, Steven clones entire repository and finds that following lines of code is not as per business requirement.

```
body {
 background-color: blue;  [Symbol] (ERROR in src code)
}
h1{
color: red;
}
```

6. Steven fixes the code as per business need.

```
body {
 background-color: powderblue;  [Symbol] (Changes made by Steven)
}
h1{
color: red;
}
```

7. Steven performs Add, Commit and Push operations to central repository.
8. John finishes his work on new business requirement. He adds the following lines of code and performs ACP.

```
body {
font-family:verdana;
font-size: 30px;
 background-color: blue;
}
h1{
```

color: red;
}

9. But while pushing the operation fails with a error msg called as “MERGE CONFLICT”.
10. John will have to manually resolve conflict and will have to do a new ACP operation.

Git Pull

git pull command first runs git fetch which downloads content from the specified remote repository. Then a git merge is executed to merge the remote content refs and heads into a new local merge commit.

Page Break

Git Stash

Suppose you are implementing a new feature for your product.
Your code is in progress and suddenly a customer escalation comes.
Because of this, you have to keep aside your new feature work for a few hours.
You cannot commit your partial code and also cannot throw away your changes.
So, you need some temporary space, where you can store your partial changes and later on commit it.

So, you need to stash your changes on to some free space. One way, is to copy-paste the code in some other folder or directory. But we already seen it in Local VCS, that how it is error prone.
Luckily, git has solution to it.

1. Buffer your half-done work
   git stash
2. Checking the buffer area
   git stash list
3. Retrieving your un-finished work from buffer
   *git stash pop **stash-id***
4. If you simply want to apply the stash,
   *git stash apply **stash-id***
5. Clearing the buffer
   git stash clear

GIT CHECKOUT

1. Using Git checkout operation to handle the error situation.
   *git checkout **filename/directory-name***
   (Helps to get back the modified/deleted file or directory in uncommitted state)
2. Using Git checkout to switch a branch in working directory.
   *git checkout **branch-name***

GIT RESET:

1. Moving the files from Staging Area to Unstaging Area
   *git reset **.** **or** git reset **file1 file2** **or** git reset*
2. Rolling back to previous commit state.
   git reset – -hard

GIT BRANCH

What are branches ?

In git we can create branches, branches are the ways to segregate the work as per
different business requirements within a project.
Branches can be given any name, you can name it as current feature in progress or it can be a bugfix branch, etc.
Default is called master branch. It is recommended to maintain the master branch neat and clean.

1. Creating a branch

   *git branch **branch-name***

   (Alternatively, we can also use the ***checkout*** to create a new branch. Eg. *git checkout -b **branch-name***)
2. Listing all the branches

   git branch -a
3. Listing only remote branches

   git branch -r
4. Renaming a branch

   git branch -m **old-branch-name** new-**branch-name**
5. Deleting branches

   git branch -d **branch-name**

   OR

   git branch -D **branch-name**

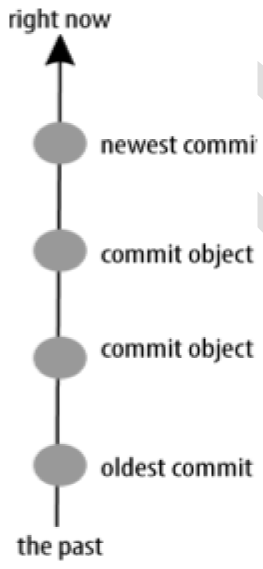
Different Branching Strategy

Selecting a perfect branching strategy is very important, mostly branching strategy differs from one industry to another.
In order, to select a branching strategy you need to work with your project manager and leads in order to decide one.
Or you may, use the existing one in practice.

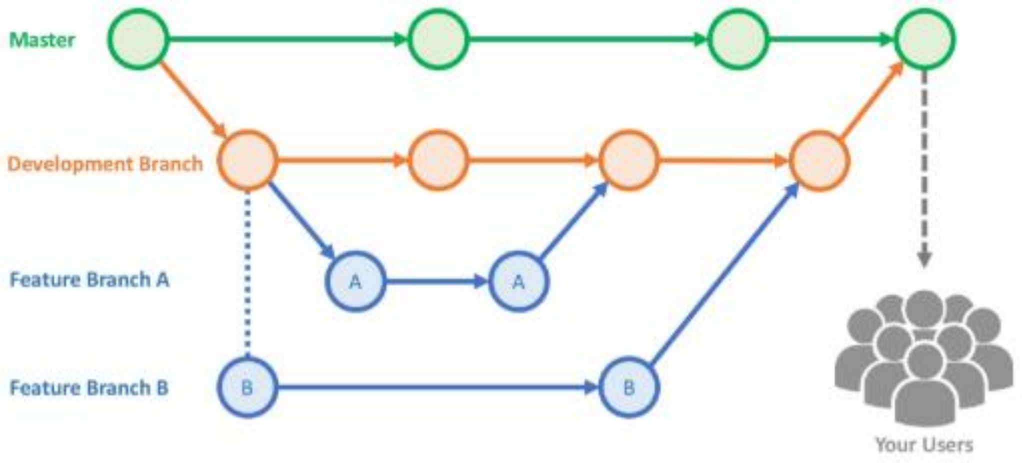
Three most common strategy adopted in industry.

- Mainline Strategy
- Feature Branching Strategy
- State Branching Strategy

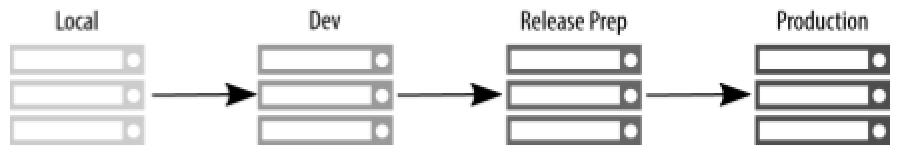
Mainline Strategy:



Feature-Branching Strategy:



State Branching Strategy


Local
Dev
Release Prep
Production