Git Commands:

-> git config

Usage:git config –global user.name "[name]"

Usage:git config –global user.email "[email address]"

This command sets the author name and email address respectively to be used with your commits.

-> **git init**

Usage: git init [repository name]

This command is used to start a new repository.

-> git clone

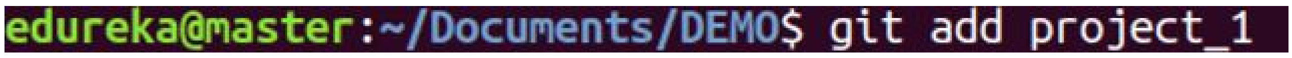
Usage:git clone [url]

This command is used to obtain a repository from an existing URL.

-> **git add**

Usage: git add [file]

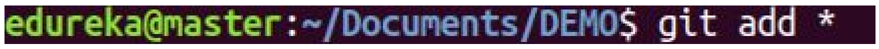
This command adds a file to the staging area.



```
edureka@master:~/Documents/DEMO$ git add project_1
```

Usage: git add *

This command adds one or more to the staging area.

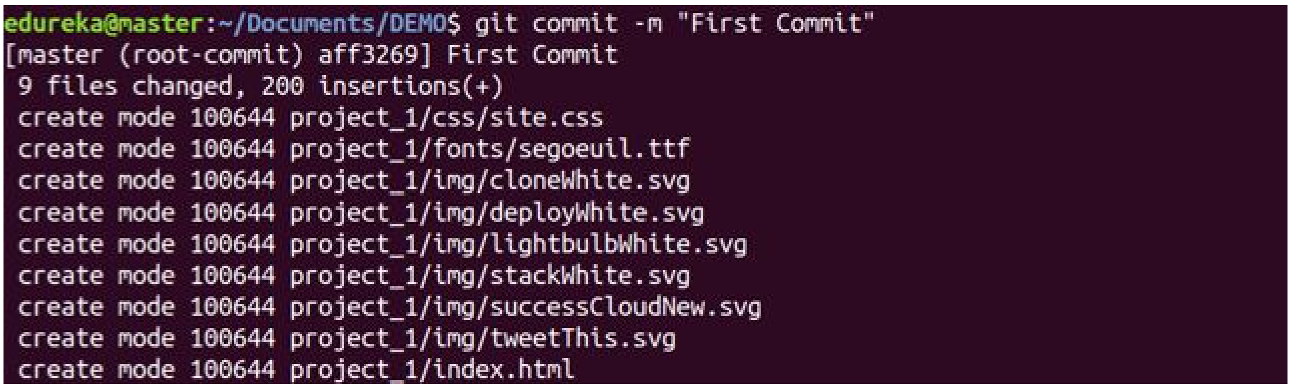


```
edureka@master:~/Documents/DEMO$ git add *
```

-> **git commit**

Usage: git commit -m "[Type in the commit message]"

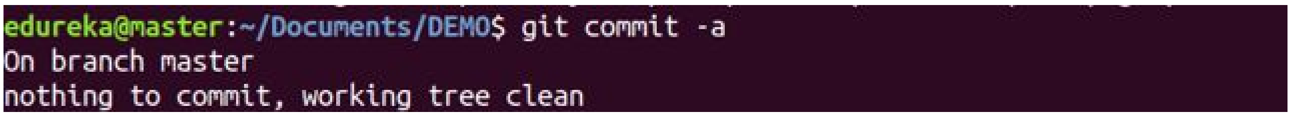
This command records or snapshots the file permanently in the version history.



```
edureka@master:~/Documents/DEMO$ git commit -m "First Commit"
[master (root-commit) aff3269] First Commit
 9 files changed, 200 insertions(+)
 create mode 100644 project_1/css/site.css
 create mode 100644 project_1/fonts/segoeuil.ttf
 create mode 100644 project_1/img/cloneWhite.svg
 create mode 100644 project_1/img/deployWhite.svg
 create mode 100644 project_1/img/lightbulbWhite.svg
 create mode 100644 project_1/img/stackWhite.svg
 create mode 100644 project_1/img/successCloudNew.svg
 create mode 100644 project_1/img/tweetThis.svg
 create mode 100644 project_1/index.html
```

Usage: **git commit -a**

This command commits any files you've added with the git add command and also commits any files you've changed since then.

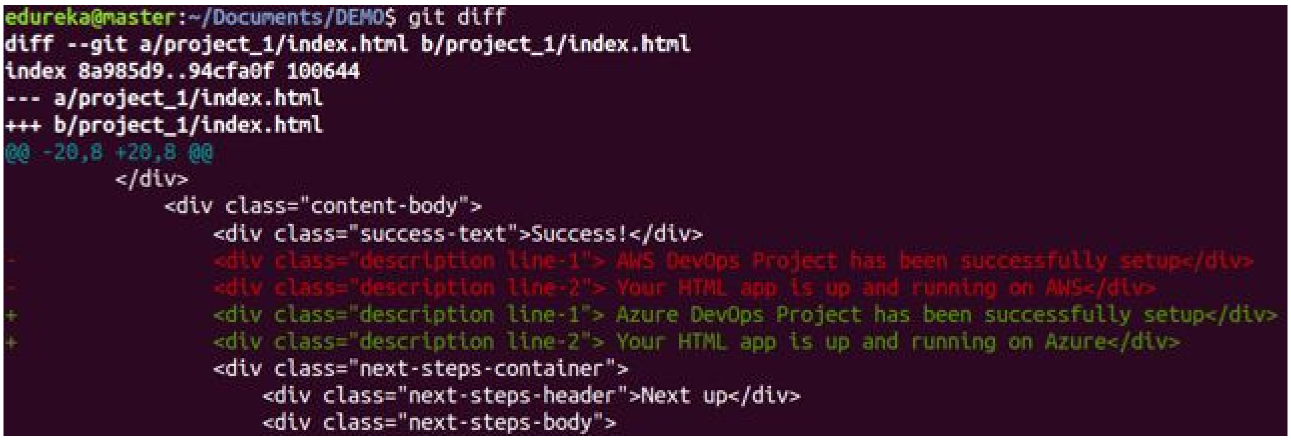


```
edureka@master:~/Documents/DEMO$ git commit -a
On branch master
nothing to commit, working tree clean
```

-> **git diff**

Usage: git diff

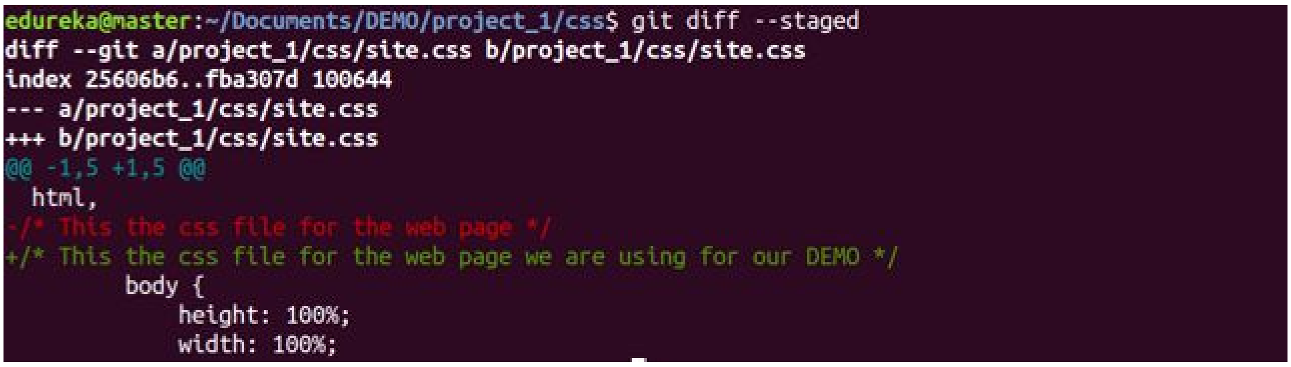
This command shows the file differences which are not yet staged.



```
edureka@master:~/Documents/DEMO$ git diff
diff --git a/project_1/index.html b/project_1/index.html
index 8a985d9..94cfa0f 100644
--- a/project_1/index.html
+++ b/project_1/index.html
@@ -20,8 +20,8 @@
         </div>
             <div class="content-body">
                 <div class="success-text">Success!</div>
-                <div class="description line-1"> AWS DevOps Project has been successfully setup</div>
-                <div class="description line-2"> Your HTML app is up and running on AWS</div>
+                <div class="description line-1"> Azure DevOps Project has been successfully setup</div>
+                <div class="description line-2"> Your HTML app is up and running on Azure</div>
                 <div class="next-steps-container">
                     <div class="next-steps-header">Next up</div>
                     <div class="next-steps-body">
```

Usage: git diff –staged

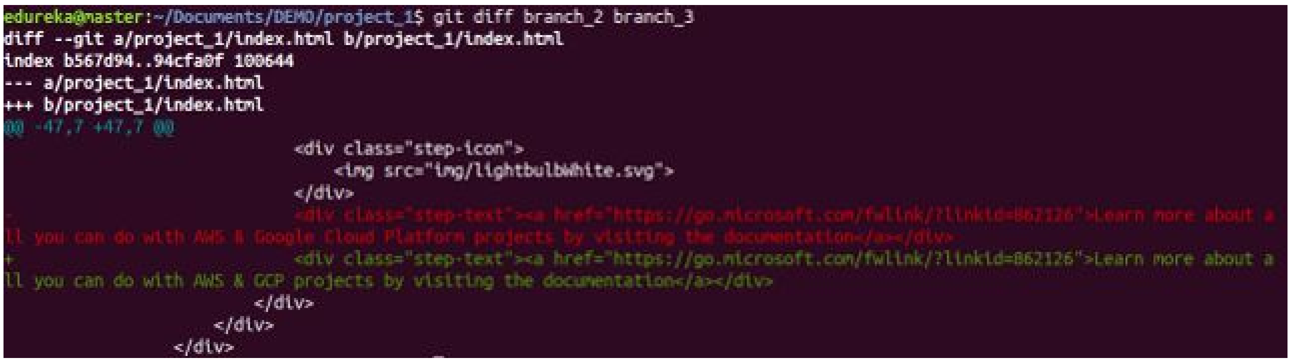
This command shows the differences between the files in the staging area and the latest version present.



```
edureka@master:~/Documents/DEMO/project_1/css$ git diff --staged
diff --git a/project_1/css/site.css b/project_1/css/site.css
index 25606b6..fba307d 100644
--- a/project_1/css/site.css
+++ b/project_1/css/site.css
@@ -1,5 +1,5 @@
  html,
-/* This the css file for the web page */
+/* This the css file for the web page we are using for our DEMO */
         body {
             height: 100%;
             width: 100%;
```

Usage:git diff [first branch] [second branch]

This command shows the differences between the two branches mentioned.

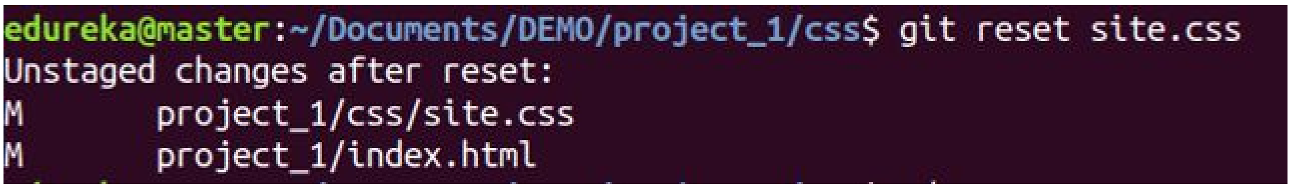


```
edureka@master:~/Documents/DEMO/project_1$ git diff branch_2 branch_3
diff --git a/project_1/index.html b/project_1/index.html
index b567d94..94cfa0f 100644
--- a/project_1/index.html
+++ b/project_1/index.html
@@ -47,7 +47,7 @@
                            <div class="step-icon">
                                <img src="img/lightbulbWhite.svg">
                            </div>
-                           <div class="step-text"><a href="https://go.microsoft.com/fwlink/?linkid=862126">Learn more about a
ll you can do with AWS & Google Cloud Platform projects by visiting the documentation</a></div>
+                           <div class="step-text"><a href="https://go.microsoft.com/fwlink/?linkid=862126">Learn more about a
ll you can do with AWS & GCP projects by visiting the documentation</a></div>
                        </div>
                    </div>
                </div>
```

-> git reset

Usage:git reset [file]

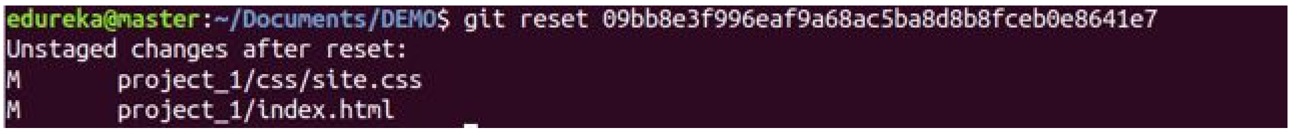
This command unstages the file, but it preserves the file contents.



```
edureka@master:~/Documents/DEMO/project_1/css$ git reset site.css
Unstaged changes after reset:
M       project_1/css/site.css
M       project_1/index.html
```

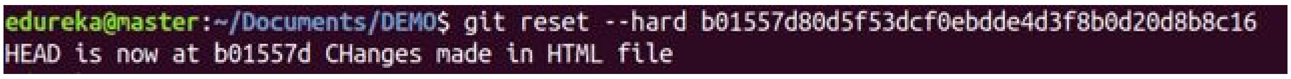
Usage:**git reset [commit]**

This command undoes all the commits after the specified commit and preserves the changes locally.



```
edureka@master:~/Documents/DEMO$ git reset 09bb8e3f996eaf9a68ac5ba8d8b8fceb0e8641e7
Unstaged changes after reset:
M       project_1/css/site.css
M       project_1/index.html
```

Usage:git reset –hard [commit]This command discards all history and goes back to the specified commit.

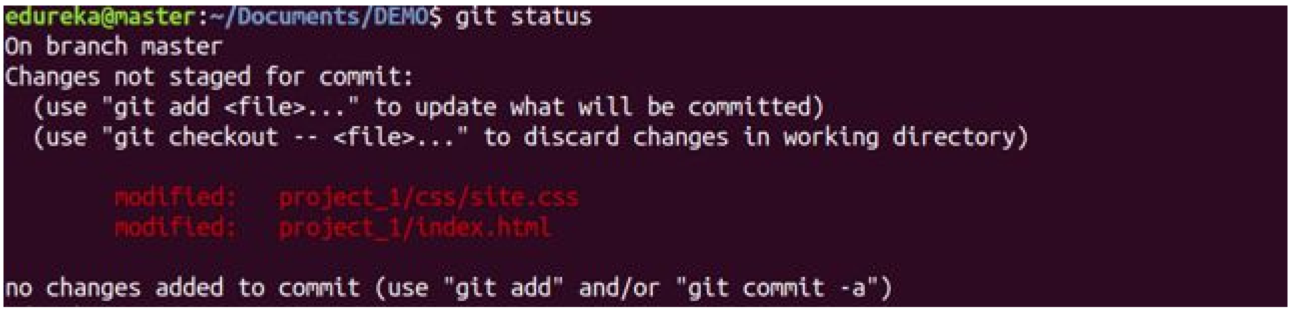


```
edureka@master:~/Documents/DEMO$ git reset --hard b01557d80d5f53dcf0ebdde4d3f8b0d20d8b8c16
HEAD is now at b01557d CHanges made in HTML file
```

-> git status

Usage:git status

This command lists all the files that have to be committed.

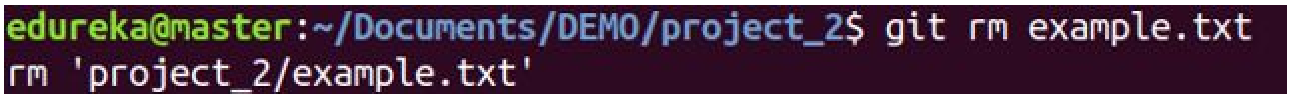


```
edureka@master:~/Documents/DEMO$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   project_1/css/site.css
        modified:   project_1/index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

-> git rm

Usage:**git rm [file]**

This command deletes the file from your working directory and stages the deletion.



```
edureka@master:~/Documents/DEMO/project_2$ git rm example.txt
rm 'project_2/example.txt'
```

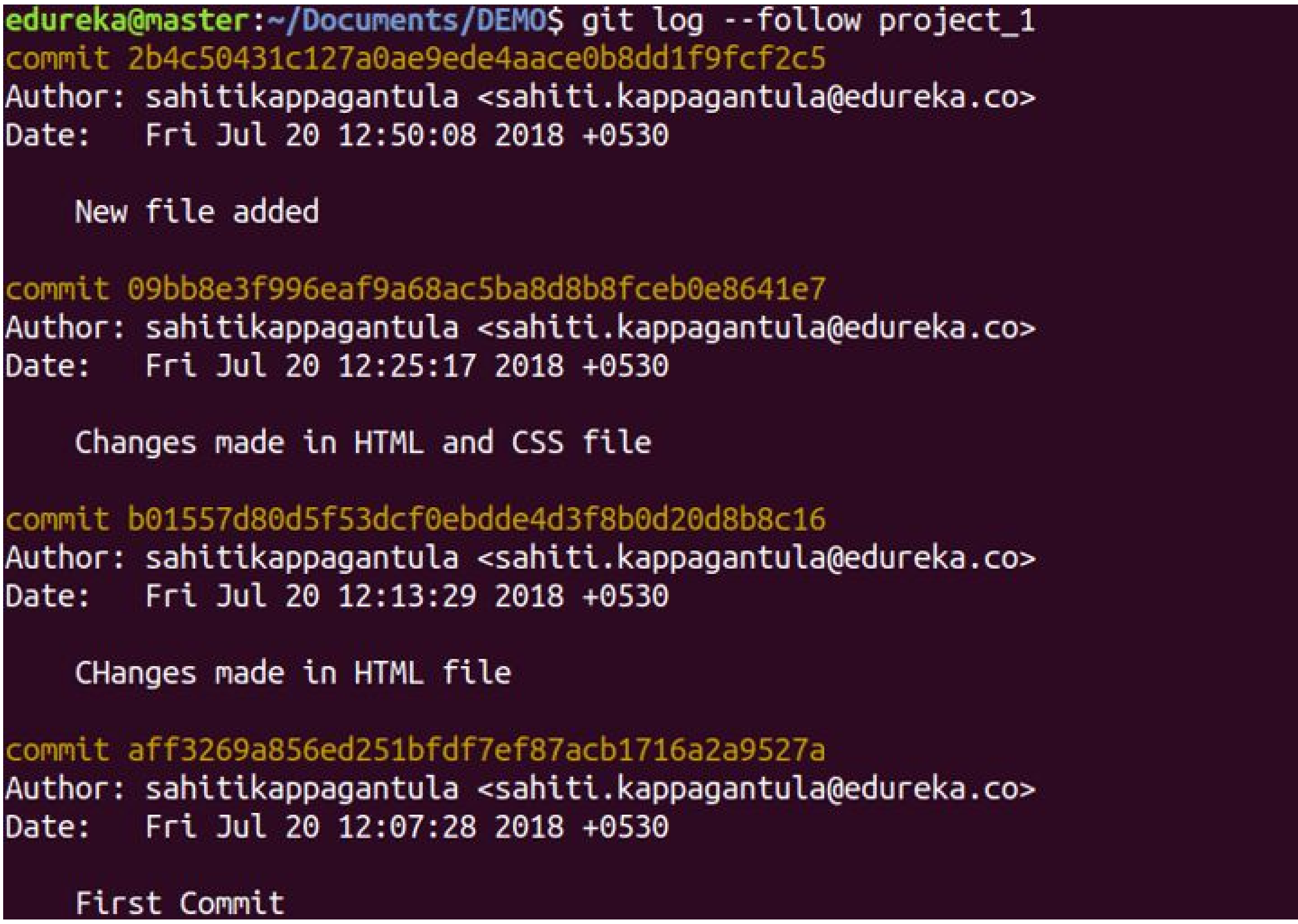
-> git log

Usage: git log

This command is used to list the version history for the current branch.

Usage:**git log –follow[file]**

This command lists version history for a file, including the renaming of files also.

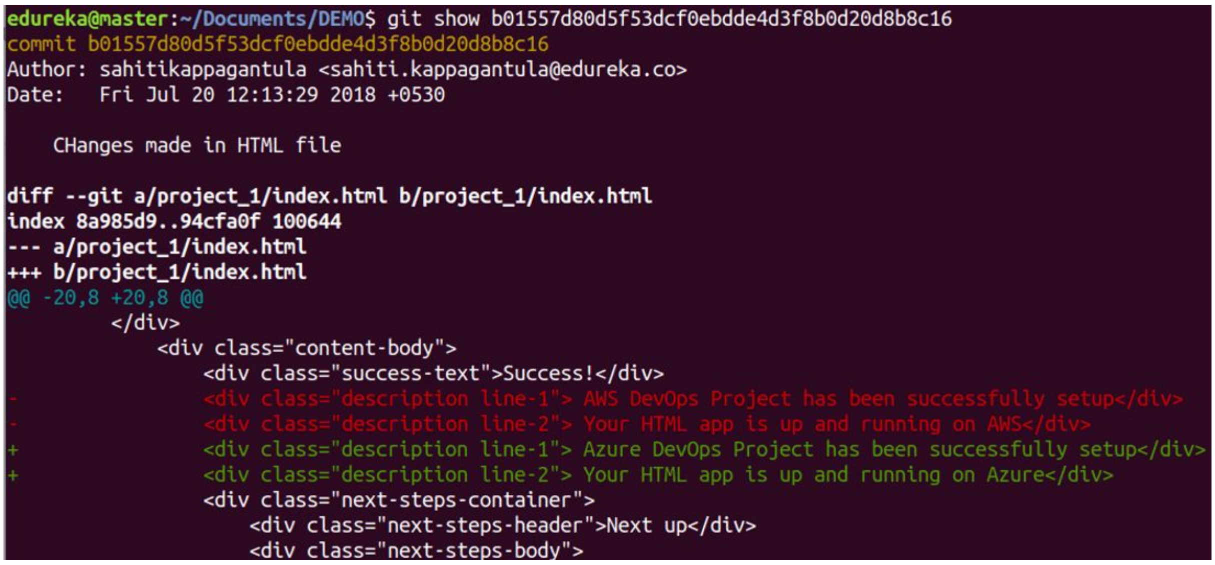


```
edureka@master:~/Documents/DEMO$ git log --follow project_1
commit 2b4c50431c127a0ae9ede4aace0b8dd1f9fcf2c5
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
Date:   Fri Jul 20 12:50:08 2018 +0530

    New file added

commit 09bb8e3f996eaf9a68ac5ba8d8b8fceb0e8641e7
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
Date:   Fri Jul 20 12:25:17 2018 +0530

    Changes made in HTML and CSS file

commit b01557d80d5f53dcf0ebdde4d3f8b0d20d8b8c16
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
Date:   Fri Jul 20 12:13:29 2018 +0530

    CHanges made in HTML file

commit aff3269a856ed251bfdf7ef87acb1716a2a9527a
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
Date:   Fri Jul 20 12:07:28 2018 +0530

    First Commit
```

-> **git show**

Usage: git show [commit]

This command shows the metadata and content changes of the specified commit.

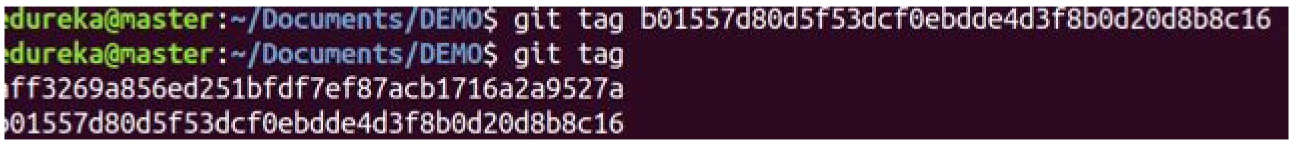


```
edureka@master:~/Documents/DEMO$ git show b01557d80d5f53dcf0ebdde4d3f8b0d20d8b8c16
commit b01557d80d5f53dcf0ebdde4d3f8b0d20d8b8c16
Author: sahitikappagantula <sahiti.kappagantula@edureka.co>
Date:   Fri Jul 20 12:13:29 2018 +0530

    CHanges made in HTML file

diff --git a/project_1/index.html b/project_1/index.html
index 8a985d9..94cfa0f 100644
--- a/project_1/index.html
+++ b/project_1/index.html
@@ -20,8 +20,8 @@
         </div>
             <div class="content-body">
                 <div class="success-text">Success!</div>
-                <div class="description line-1"> AWS DevOps Project has been successfully setup</div>
-                <div class="description line-2"> Your HTML app is up and running on AWS</div>
+                <div class="description line-1"> Azure DevOps Project has been successfully setup</div>
+                <div class="description line-2"> Your HTML app is up and running on Azure</div>
                 <div class="next-steps-container">
                     <div class="next-steps-header">Next up</div>
                     <div class="next-steps-body">
```

git tag

Usage: **git tag [commitID]**

This command is used to give tags to the specified commit.

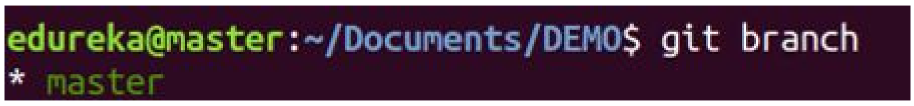


```
dureka@master:~/Documents/DEMO$ git tag b01557d80d5f53dcf0ebdde4d3f8b0d20d8b8c16
dureka@master:~/Documents/DEMO$ git tag
ff3269a856ed251bfdf7ef87acb1716a2a9527a
01557d80d5f53dcf0ebdde4d3f8b0d20d8b8c16
```

-> **git branch**

Usage: git branch

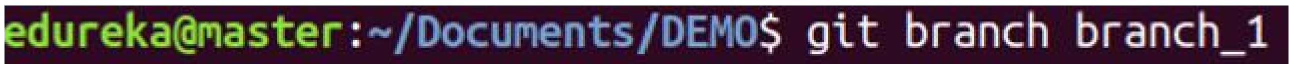
This command lists all the local branches in the current repository.



```
edureka@master:~/Documents/DEMO$ git branch
* master
```

Usage: git branch [branch name]

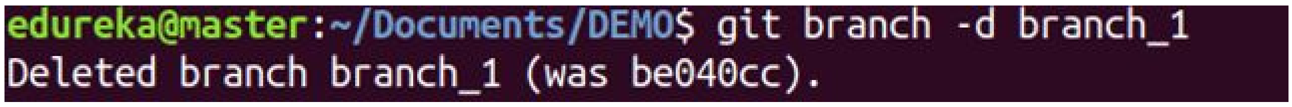
This command creates a new branch.



```
edureka@master:~/Documents/DEMO$ git branch branch_1
```

Usage: git branch -d [branch name]

This command deletes the feature branch.

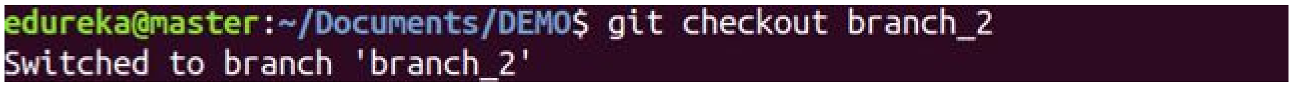


```
edureka@master:~/Documents/DEMO$ git branch -d branch_1
Deleted branch branch_1 (was be040cc).
```

-> **git checkout**

Usage:git checkout [branch name]

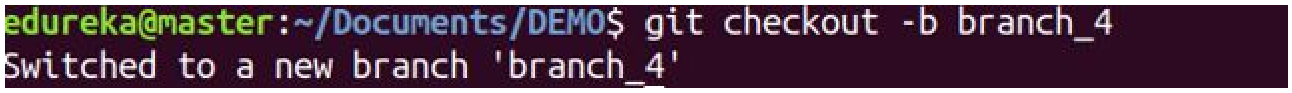
This command is used to switch from one branch to another.



```
edureka@master:~/Documents/DEMO$ git checkout branch_2
Switched to branch 'branch_2'
```

Usage:git checkout -b [branch name]

This command creates a new branch and also switches to it.

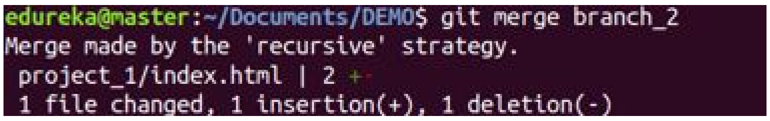


```
edureka@master:~/Documents/DEMO$ git checkout -b branch_4
Switched to a new branch 'branch_4'
```

-> git merge

Usage: git merge [branch name]

This command merges the specified branch's history into the current branch.

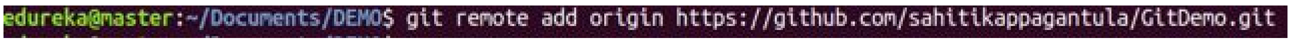


```
edureka@master:~/Documents/DEMO$ git merge branch_2
Merge made by the 'recursive' strategy.
 project_1/index.html | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

-> git remote

Usage: git remote add [variable name] [Remote Server Link]

This command is used to connect your local repository to the remote server.

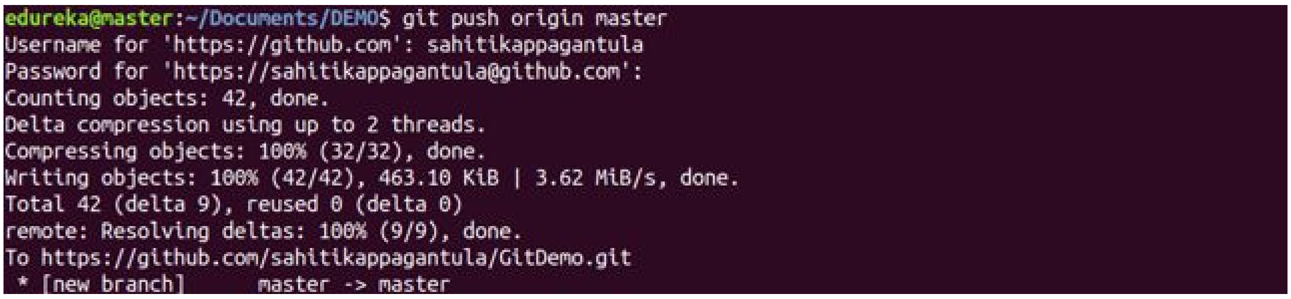


```
edureka@master:~/Documents/DEMO$ git remote add origin https://github.com/sahitikappagantula/GitDemo.git
```

-> git push

Usage: git push [variable name] master

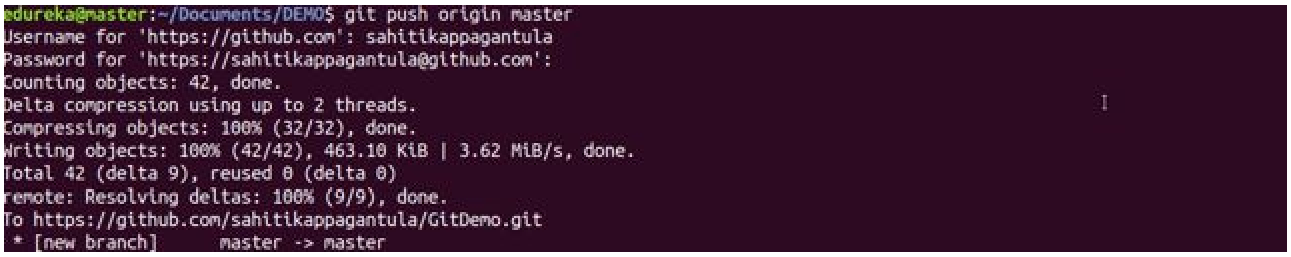
This command sends the committed changes of master branch to your remote repository.



```
edureka@master:~/Documents/DEMO$ git push origin master
Username for 'https://github.com': sahitikappagantula
Password for 'https://sahitikappagantula@github.com':
Counting objects: 42, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (32/32), done.
Writing objects: 100% (42/42), 463.10 KiB | 3.62 MiB/s, done.
Total 42 (delta 9), reused 0 (delta 0)
remote: Resolving deltas: 100% (9/9), done.
To https://github.com/sahitikappagantula/GitDemo.git
 * [new branch]      master -> master
```

Usage: git push [variable name] [branch]

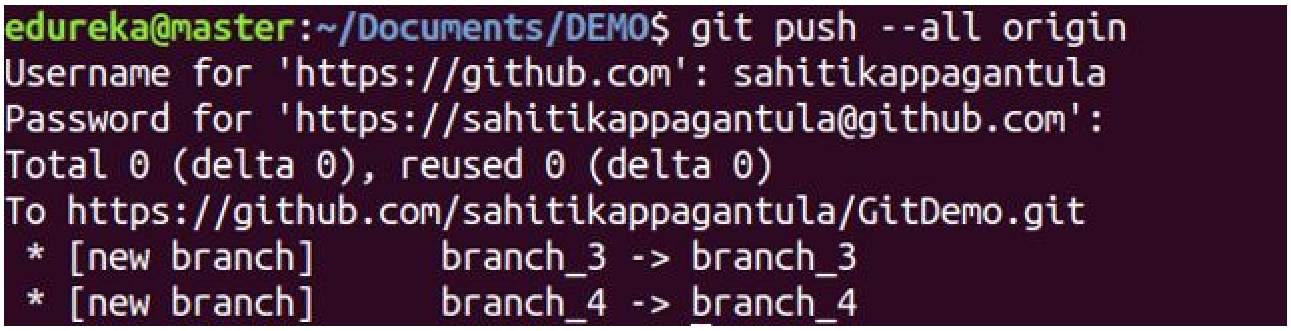
This command sends the branch commits to your remote repository.



```
edureka@master:~/Documents/DEMO$ git push origin master
Username for 'https://github.com': sahitikappagantula
Password for 'https://sahitikappagantula@github.com':
Counting objects: 42, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (32/32), done.
Writing objects: 100% (42/42), 463.10 KiB | 3.62 MiB/s, done.
Total 42 (delta 9), reused 0 (delta 0)
remote: Resolving deltas: 100% (9/9), done.
To https://github.com/sahitikappagantula/GitDemo.git
 * [new branch]      master -> master
```

Usage: git push –all [variable name]

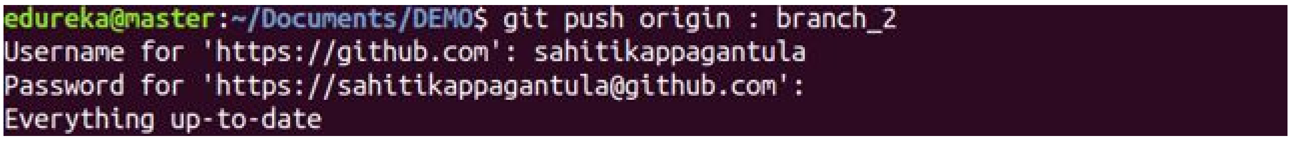
This command pushes all branches to your remote repository.



```
edureka@master:~/Documents/DEMO$ git push --all origin
Username for 'https://github.com': sahitikappagantula
Password for 'https://sahitikappagantula@github.com':
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/sahitikappagantula/GitDemo.git
 * [new branch]      branch_3 -> branch_3
 * [new branch]      branch_4 -> branch_4
```

Usage: git push [variable name] :[branch name]

This command deletes a branch on your remote repository.



```
edureka@master:~/Documents/DEMO$ git push origin : branch_2
Username for 'https://github.com': sahitikappagantula
Password for 'https://sahitikappagantula@github.com':
Everything up-to-date
```

-> git pull

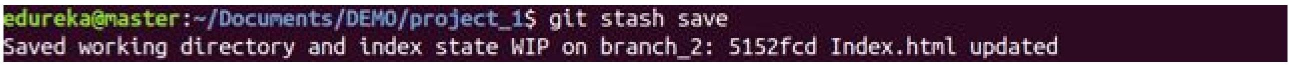
Usage: git pull [Repository Link]

This command fetches and merges changes on the remote server to your working directory.

-> git stash

Usage: git stash save

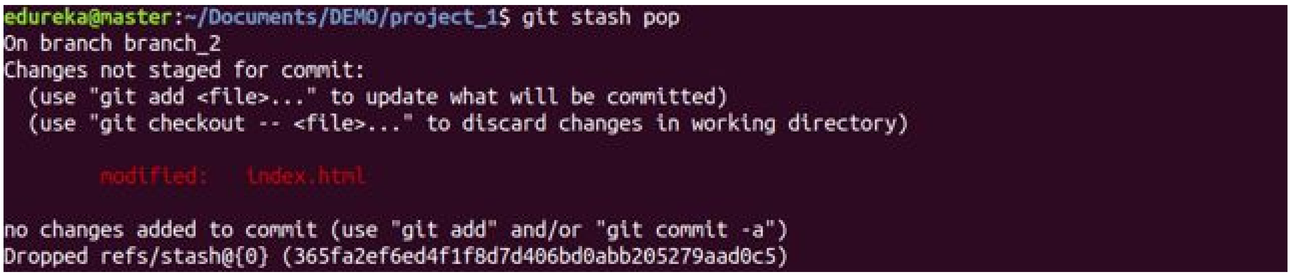
This command temporarily stores all the modified tracked files.



```
edureka@master:~/Documents/DEMO/project_1$ git stash save
Saved working directory and index state WIP on branch_2: 5152fcd Index.html updated
```

Usage: git stash pop

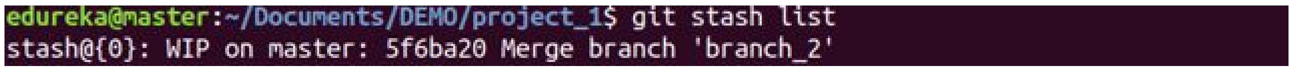
This command restores the most recently stashed files.



```
edureka@master:~/Documents/DEMO/project_1$ git stash pop
On branch branch_2
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (365fa2ef6ed4f1f8d7d406bd0abb205279aad0c5)
```

Usage: git stash list

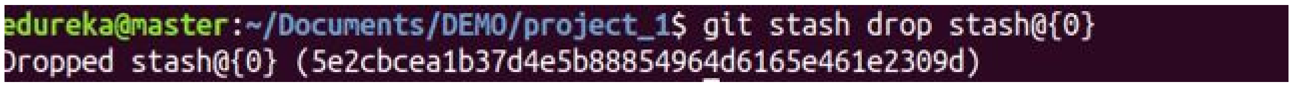
This command lists all stashed changesets.



```
edureka@master:~/Documents/DEMO/project_1$ git stash list
stash@{0}: WIP on master: 5f6ba20 Merge branch 'branch_2'
```

Usage: git stash drop

This command discards the most recently stashed changeset.



```
edureka@master:~/Documents/DEMO/project_1$ git stash drop stash@{0}
Dropped stash@{0} (5e2cbcea1b37d4e5b88854964d6165e461e2309d)
```

Git Log Commands

1. Display All Commits

You can force the log tool display all commits (regardless of the branch checked out) by using the –all option.

```
$ git log --all

commit c36d2103222cfd9ad62f755fee16b3f256f1cb21

Author: Bob Smith <BSmith@example.com>

Date:   Tue Mar 25 22:09:26 2014 -0300



  Ut sit.



commit 97eda7d2dab729eda23eefdc14336a5644e3c748

Author: John Doe <JDoe@example.com>

Date:   Mon Mar 24 10:14:08 2014 -0300
```

```
  Mollis interdum ullamcorper sociosqu, habitasse arcu magna risus congue dictum arcu, odio.

.

.

.
```

2. View n Most Recent Commits

The real power of the Git Log tool, however, is in its diversity. There are many options that not only allow you to filter commits to almost any granularity you desire, but to also tailor the format of the output to you personal needs.

The most trivial way to filter commits is to limit output to the ‘n’ most recently committed ones. This can be accomplished using the -<n> option. Replace <n> with the number of commits you would like to see. i.e. To see the 3 most recent commits:

```
$ git log -3

commit c36d2103222cfd9ad62f755fee16b3f256f1cb21

Author: Bob Smith <BSmith@example.com>

Date:   Tue Mar 25 22:09:26 2014 -0300



  Ut sit.



commit 97eda7d2dab729eda23eefdc14336a5644e3c748

Author: John Doe <JDoe@example.com>

Date:   Mon Mar 24 10:14:08 2014 -0300



  Mollis interdum ullamcorper sociosqu, habitasse arcu magna risus congue dictum arcu, odio.
```

```
commit 3ca28cfa2b8ea0d765e808cc565e056a94aceaf5

Author: Bobby Jones <BJones@example.com>

Date:   Mon Mar 24 01:52:04 2014 -0300



  Fermentum magnis facilisis torquent platea sapien hac, aliquet torquent ad netus
risus.
```

3. Filter Commits By Author or Committer

Another common way to filter commits is by the person who wrote or committed the changes. This can be done using the –author and –committer options. The syntax is

```
git log --author <name>

git log --committer <name>
```

The author option will limit results to commits in which the changes were written by <name>. The –committer option, on the other hand, will limit results to commits that were committed by that individual. Many times, the author and committer will be the same person (you would generally expect this to be the case) but, in the case where a developer submits a patch of their work for approval, the developer may not actually commit the code.

```
$ git log --author=Bob

commit c36d2103222cfd9ad62f755fee16b3f256f1cb21

Author: Bob Smith <BSmith@example.com>

Date:   Tue Mar 25 22:09:26 2014 -0300



  Ut sit.



commit 3ca28cfa2b8ea0d765e808cc565e056a94aceaf5

Author: Bobby Jones <BJones@example.com>
```

```
Date:   Mon Mar 24 01:52:04 2014 -0300



  Fermentum magnis facilisis torquent platea sapien hac, aliquet torquent ad netus
risus.



commit cfc101ad280f5b005c8d49c91e849c6c40a1d275

Author: Bob Smith <BSmith@example.com>

Date:   Thu Mar 20 10:31:22 2014 -0300



  Natoque, turpis per vestibulum neque nibh ullamcorper.

.

.

.
```

Note: Notice how this option matches any commit in which the <name> we specify is a substring match of the commit's author (We've found both Bob Smith's and Bobby Jones' commits).

4. Filter Commits by X Days Ago

Many times you'll want to limit the commits to those within a given date range. This can be accomplished using the –before and –after options:

```
git log --before <date>

git log --after <date>
```

The date can be specified as a string with the format: "yyyy-mm-dd". Git will also accept Ruby expressions as arguments here, so you can do things like the following to see commits that occurred in the last 2 days

```
$ git log --after 2.days.ago

commit c36d2103222cfd9ad62f755fee16b3f256f1cb21
```

```
Author: Bob Smith <BSmith@example.com>

Date:   Tue Mar 25 22:09:26 2014 -0300



  Ut sit.



commit 97eda7d2dab729eda23eefdc14336a5644e3c748

Author: John Doe <JDoe@example.com>

Date:   Mon Mar 24 10:14:08 2014 -0300



  Mollis interdum ullamcorper sociosqu, habitasse arcu magna risus congue dictum arcu,
odio.



commit 3ca28cfa2b8ea0d765e808cc565e056a94aceaf5

Author: Bobby Jones <BJones@example.com>

Date:   Mon Mar 24 01:52:04 2014 -0300



  Fermentum magnis facilisis torquent platea sapien hac, aliquet torquent ad netus
risus.
```

5. Filter Commits by Date Range

To specify a date range, use both options:

```
git log --after <date> --before <date>
```

To see the commits that occurred on Feb 2nd, 2014:

```
$ git log --after "2014-02-01" --before "2014-02-02"
```

```
commit 69e1684ae9605544707fc36a7bf37da93dc7b015

Author: Bob Smith <BSmith@example.com>

Date:   Sun Feb 2 01:26:00 2014 -0400



  Praesent tempus varius vel feugiat mi tempor felis parturient.
```

6. View All Diff of Changes for Each Commit

Options to Modifying format of the output: To view the entire diff of changes for each commit found, use the -p option (think ‘p’ for patch):

```
$ git log -p

commit c36d2103222cfd9ad62f755fee16b3f256f1cb21

Author: Bob Smith <BSmith@example.com>

Date:   Tue Mar 25 22:09:26 2014 -0300



  Ut sit.



  diff --git a/foo.txt b/foo.txt

  index 5554f5b..2773ba4 100644

  --- a/foo.txt

  +++ b/foo.txt

  @@ -436,3 +436,4 @@ Fermentum mollis.

   Lacus fermentum nonummy purus amet aliquam taciti fusce facilisis magna.

    Viverra facilisi curae augue.

     Purus ve nunc mi consectetuer cras.
    +Ad, maecenas egestas viverra blandit odio.



commit 97eda7d2dab729eda23eefdc14336a5644e3c748

Author: John Doe <JDoe@example.com>

Date:   Mon Mar 24 10:14:08 2014 -0300



  Mollis interdum ullamcorper sociosqu, habitasse arcu magna risus congue dictum arcu,
odio.



  diff --git a/foo.txt b/foo.txt

  index 9cdef98..5554f5b 100644

  --- a/foo.txt

  +++ b/foo.txt

  @@ -435,3 +435,4 @@ Lacinia et enim suspendisse conubia lacus.

   Fermentum mollis.

    Lacus fermentum nonummy purus amet aliquam taciti fusce facilisis magna.

     Viverra facilisi curae augue.

    +Purus ve nunc mi consectetuer cras.

.

.

.
```

7. View Summary of Changes for Each Commit

To view a summary of the changes made in each commit (# of lines added, removed, etc), use the –stat option:

```
$ git log --stat

commit c36d2103222cfd9ad62f755fee16b3f256f1cb21

Author: Bob Smith <BSmith@example.com>

Date:   Tue Mar 25 22:09:26 2014 -0300



  Ut sit.



   foo.txt | 1 +

    1 file changed, 1 insertion(+)



commit 97eda7d2dab729eda23eefdc14336a5644e3c748

Author: John Doe <JDoe@example.com>

Date:   Mon Mar 24 10:14:08 2014 -0300



  Mollis interdum ullamcorper sociosqu, habitasse arcu magna risus congue dictum arcu,
odio.



   foo.txt | 1 +

    1 file changed, 1 insertion(+)
```

8. View Just One Line Per Commit

To just get the bare minimum information in a single line per commit, use the –oneline option. Each commit will be shown as simply the commit hash followed by the commit message, on a single line:

```
$ git log --oneline
```

```
c36d210 Ut sit.

97eda7d Mollis interdum ullamcorper sociosqu, habitasse arcu magna risus congue dictum
arcu, odio.

3ca28cf Fermentum magnis facilisis torquent platea sapien hac, aliquet torquent ad
netus risus.

3a96c1e Proin aenean vestibulum sociosqu vitae platea, odio, nisi habitasse at, in
lorem odio varius.

1f0548c Nulla odio feugiat, id, volutpat litora, adipiscing.

cfc101a Natoque, turpis per vestibulum neque nibh ullamcorper.

.

.

.
```

9. View Commit History in ASCII Graph

The Git Log tool can also display the commit history in an ascii art graphical representation with the –graph option. This option works well when combined with the –oneline option mentioned above.

```
$ git log --graph

* commit c36d2103222cfd9ad62f755fee16b3f256f1cb21

| Author: Bob Smith <BSmith@example.com>

| Date:   Tue Mar 25 22:09:26 2014 -0300

|

|     Ut sit.

|

* commit 97eda7d2dab729eda23eefdc14336a5644e3c748

| Author: John Doe <JDoe@example.com>

| Date:   Mon Mar 24 10:14:08 2014 -0300
```

```
|

|     Mollis interdum ullamcorper sociosqu, habitasse arcu magna risus congue dictum arcu, odio.

|

* commit 3ca28cfa2b8ea0d765e808cc565e056a94aceaf5

| Author: Bobby Jones <BJones@example.com>

| Date:   Mon Mar 24 01:52:04 2014 -0300

|

|     Fermentum magnis facilisis torquent platea sapien hac, aliquet torquent ad netus risus.



.

.

.
```

10. Format the Git Log Output

To take complete control over the format of the output, use the –pretty option. This can be extremely useful if you are using the output of the log tool for further reporting. The syntax of this option is:

```
git log --pretty=format:"<options>"
```

The <options> are specified in a similar way as formatted strings are in many languages. For example:

```
$ git log --pretty=format:"Commit Hash: %H, Author: %aN, Date: %aD"

Commit Hash: c36d2103222cfd9ad62f755fee16b3f256f1cb21, Author: Bob Smith, Date: Tue, 25 Mar 2014 22:09:26 -0300

Commit Hash: 97eda7d2dab729eda23eefdc14336a5644e3c748, Author: John Doe, Date: Mon, 24 Mar 2014 10:14:08 -0300
Commit Hash: 3ca28cfa2b8ea0d765e808cc565e056a94aceaf5, Author: Bobby Jones, Date: Mon,
24 Mar 2014 01:52:04 -0300


Commit Hash: 3a96c1ed29e85f1a119ad39033511413aad616d1, Author: John Doe, Date: Sun, 23
Mar 2014 06:05:49 -0300


Commit Hash: 1f0548cc700988903380b8ca40fd1fecfa50347a, Author: John Doe, Date: Fri, 21
Mar 2014 17:53:49 -0300


.


.


.
```

SDLC

Stages:

1. Planning
2. Analysis
3. Design
4. Implementation
5. Testing and integration
6. Maintainence

Models:

1. Waterfall Model
2. RAD Model
3. Spiral Model
4. V-Model
5. Incremental Model
6. Agile Model
7. Iterative Model
8. BigBang Model

1. Waterfall Model

Winston Royce introduced the Waterfall Model in 1970.This model has five phases: Requirements analysis and specification, design, implementation, and unit testing, integration and system testing, and operation and maintenance. The steps always follow in this order and do not overlap. The developer must complete every phase before the next phase begins.

When to use SDLC Waterfall Model?

Some Circumstances where the use of the Waterfall model is most suited are:

- When the requirements are constant and not changed regularly.
- A project is short
- The situation is calm
- Where the tools and technology used is consistent and is not changing
- When resources are well prepared and are available to use.

Advantages of Waterfall model

- This model is simple to implement also the number of resources that are required for it is minimal.
- The requirements are simple and explicitly declared; they remain unchanged during the entire project development.
- The start and end points for each phase is fixed, which makes it easy to cover progress.
- The release date for the complete product, as well as its final cost, can be determined before development.
- It gives easy to control and clarity for the customer due to a strict reporting system.

Disadvantages of Waterfall model

- In this model, the risk factor is higher, so this model is not suitable for more significant and complex projects.
- This model cannot accept the changes in requirements during development.
- It becomes tough to go back to the phase. For example, if the application has now shifted to the coding phase, and there is a change in requirement, It becomes tough to go back and change it.
- Since the testing done at a later stage, it does not allow identifying the challenges and risks in the earlier phase, so the risk reduction strategy is difficult to prepare.

2. RAD (Rapid Application Development) Model

RAD is a linear sequential software development process model that emphasizes a concise development cycle using an element based construction approach. If the requirements are well understood and described, and the project scope is a constraint, the RAD process enables a development team to create a fully functional system within a concise time period.

RAD (Rapid Application Development) is a concept that products can be developed faster and of higher quality through:

- Gathering requirements using workshops or focus groups
- Prototyping and early, reiterative user testing of designs
- The re-use of software components
- A rigidly paced schedule that refers design improvements to the next product version
- Less formality in reviews and other team communication

The various phases of RAD are as follows:

1.Business Modelling: The information flow among business functions is defined by answering questions like what data drives the business process, what data is generated, who generates it, where does the information go, who process it and so on.

2. Data Modelling: The data collected from business modeling is refined into a set of data objects (entities) that are needed to support the business. The attributes (character of each entity) are identified, and the relation between these data objects (entities) is defined.

3. Process Modelling: The information object defined in the data modeling phase are transformed to achieve the data flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.

4. Application Generation: Automated tools are used to facilitate construction of the software; even they use the 4th GL techniques.

5. Testing & Turnover: Many of the programming components have already been tested since RAD emphasis reuse. This reduces the overall testing time. But the new part must be tested, and all interfaces must be fully exercised.

When to use RAD Model?

- When the system should need to create the project that modularizes in a short span time (2-3 months).
- When the requirements are well-known.
- When the technical risk is limited.
- When there's a necessity to make a system, which modularized in 2-3 months of period.
- It should be used only if the budget allows the use of automatic code generating tools.

Advantage of RAD Model

- This model is flexible for change.
- In this model, changes are adoptable.
- Each phase in RAD brings highest priority functionality to the customer.

- It reduced development time.
- It increases the reusability of features.

Disadvantage of RAD Model

- It required highly skilled designers.
- All application is not compatible with RAD.
- For smaller projects, we cannot use the RAD model.
- On the high technical risk, it's not suitable.
- Required user involvement.

3. Spiral Model

The spiral model, initially proposed by Boehm, is an evolutionary software process model that couples the iterative feature of prototyping with the controlled and systematic aspects of the linear sequential model. It implements the potential for rapid development of new versions of the software. Using the spiral model, the software is developed in a series of incremental releases. During the early iterations, the additional release may be a paper model or prototype. During later iterations, more and more complete versions of the engineered system are produced.

The Spiral Model is shown in fig:

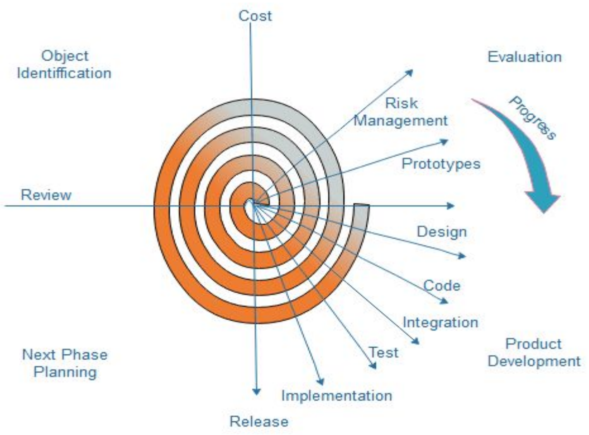


Fig. Spiral Model

Each cycle in the spiral is divided into four parts:

Objective setting: Each cycle in the spiral starts with the identification of purpose for that cycle, the various alternatives that are possible for achieving the targets, and the constraints that exists.

Risk Assessment and reduction: The next phase in the cycle is to calculate these various alternatives based on the goals and constraints. The focus of evaluation in this stage is located on the risk perception for the project.

Development and validation: The next phase is to develop strategies that resolve uncertainties and risks. This process may include activities such as benchmarking, simulation, and prototyping.

Planning: Finally, the next step is planned. The project is reviewed, and a choice made whether to continue with a further period of the spiral. If it is determined to keep, plans are drawn up for the next step of the project.

The development phase depends on the remaining risks. For example, if performance or user-interface risks are treated more essential than the program development risks, the next phase may be an evolutionary development that includes developing a more detailed prototype for solving the risks.

The risk-driven feature of the spiral model allows it to accommodate any mixture of a specification-oriented, prototype-oriented, simulation-oriented, or another type of approach. An essential element of the model is that each period of the spiral is completed by a review that includes all the products developed during that cycle, including plans for the next cycle. The spiral model works for development as well as enhancement projects.

When to use Spiral Model?

- When deliverance is required to be frequent.
- When the project is large
- When requirements are unclear and complex
- When changes may require at any time
- Large and high budget projects

Advantages

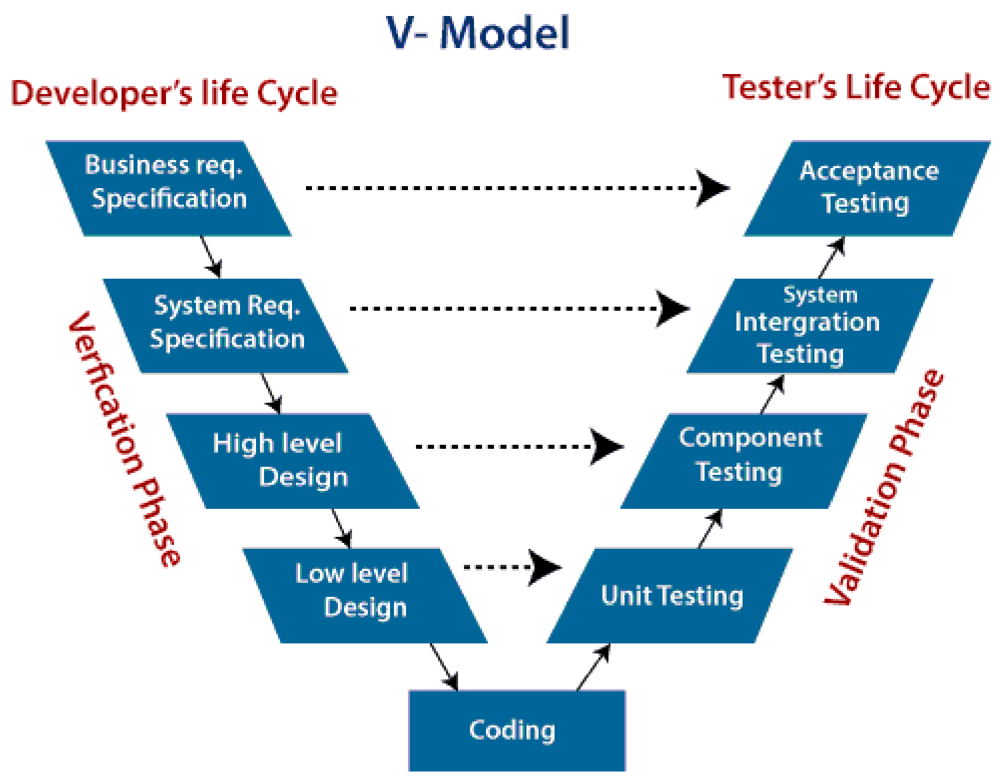
- High amount of risk analysis
- Useful for large and mission-critical projects.

Disadvantages

- Can be a costly model to use.
- Risk analysis needed highly particular expertise
- Doesn't work well for smaller projects.

4. V-Model

V-Model also referred to as the Verification and Validation Model. In this, each phase of SDLC must complete before the next phase starts. It follows a sequential design process same as the waterfall model. Testing of the device is planned in parallel with a corresponding stage of development.



Verification: It involves a static analysis method (review) done without executing code. It is the process of evaluation of the product development process to find whether specified requirements meet.

Validation: It involves dynamic analysis method (functional, non-functional), testing is done by executing code. Validation is the process to classify the software after the completion of the development process to determine whether the software meets the customer expectations and requirements.

So V-Model contains Verification phases on one side of the Validation phases on the other side. Verification and Validation process is joined by coding phase in V-shape. Thus it is known as V-Model.

There are the various phases of Verification Phase of V-model:

1. Business requirement analysis: This is the first step where product requirements understood from the customer's side. This phase contains detailed communication to understand customer's expectations and exact requirements.

2. System Design: In this stage system engineers analyze and interpret the business of the proposed system by studying the user requirements document.

3. Architecture Design: The baseline in selecting the architecture is that it should understand all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology detail, etc. The integration testing model is carried out in a particular phase.

4. Module Design: In the module design phase, the system breaks down into small modules. The detailed design of the modules is specified, which is known as Low-Level Design

5. Coding Phase: After designing, the coding phase is started. Based on the requirements, a suitable programming language is decided. There are some guidelines and standards for coding. Before checking in the repository, the final build is optimized for better performance, and the code goes through many code reviews to check the performance.

There are the various phases of Validation Phase of V-model:

1. Unit Testing: In the V-Model, Unit Test Plans (UTPs) are developed during the module design phase. These UTPs are executed to eliminate errors at code level or unit level. A unit is the smallest entity which can independently exist, e.g., a program module. Unit testing verifies that the smallest entity can function correctly when isolated from the rest of the codes/ units.

2. Integration Testing: Integration Test Plans are developed during the Architectural Design Phase. These tests verify that groups created and tested independently can coexist and communicate among themselves.

3. System Testing: System Tests Plans are developed during System Design Phase. Unlike Unit and Integration Test Plans, System Tests Plans are composed by the client?s business team. System Test ensures that expectations from an application developer are met.

4. Acceptance Testing: Acceptance testing is related to the business requirement analysis part. It includes testing the software product in user atmosphere. Acceptance tests reveal the compatibility problems with the different systems, which is available within the user atmosphere. It conjointly discovers the non-functional problems like load and performance defects within the real user atmosphere.

When to use V-Model?

 When the requirement is well defined and not ambiguous.

 The V-shaped model should be used for small to medium-sized projects where requirements are clearly defined and fixed.

 The V-shaped model should be chosen when sample technical resources are available with essential technical expertise.

Advantage (Pros) of V-Model:

1. Easy to Understand.
2. Testing Methods like planning, test designing happens well before coding.
3. This saves a lot of time. Hence a higher chance of success over the waterfall model.
4. Avoids the downward flow of the defects.
5. Works well for small plans where requirements are easily understood.

Disadvantage (Cons) of V-Model:

1. Very rigid and least flexible.
2. Not a good for a complex project.
3. Software is developed during the implementation stage, so no early prototypes of the software are produced.
4. If any changes happen in the midway, then the test documents along with the required documents, has to be updated.

5. Incremental Model

Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle. In this model, each module goes through the requirements, design, implementation and testing phases. Every subsequent release of the module adds function to the previous release. The process continues until the complete system achieved.

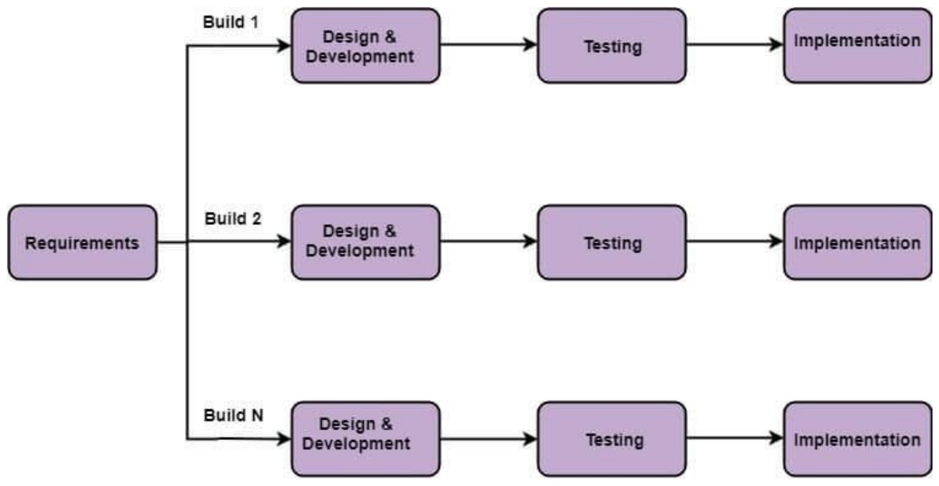


Fig: Incremental Model

The various phases of incremental model are as follows:

1. Requirement analysis: In the first phase of the incremental model, the product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team. To develop the software under the incremental model, this phase performs a crucial role.

2. Design & Development: In this phase of the Incremental model of SDLC, the design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.

3. Testing: In the incremental model, the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.

4. Implementation: Implementation phase enables the coding phase of the development system. It involves the final coding that design in the designing and development phase and tests the functionality in the testing phase. After completion of this phase, the number of the product working is enhanced and upgraded up to the final system product

When we use the Incremental Model?

- When the requirements are superior.
- A project has a lengthy development schedule.
- When Software team are not very well skilled or trained.
- When the customer demands a quick release of the product.
- You can develop prioritized requirements first.

Advantage of Incremental Model

- Errors are easy to be recognized.
- Easier to test and debug
- More flexible.
- Simple to manage risk because it handled during its iteration.
- The Client gets important functionality early.

Disadvantage of Incremental Model

- Need for good planning
- Total Cost is high.
- Well defined module interfaces are needed.

6. Agile Model

The meaning of Agile is swift or versatile."Agile process model" refers to a software development approach based on iterative development. Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning. The project scope and requirements are laid down at the beginning of the development process. Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance.

Each iteration is considered as a short time "frame" in the Agile process model, which typically lasts from one to four weeks. The division of the entire project into smaller parts helps to minimize the project risk and to reduce the overall project delivery time requirements. Each iteration involves a team working through a full software development life cycle including planning, requirements analysis, design, coding, and testing before a working product is demonstrated to the client.

Problem with the waterfall model is: Let us take an example of a banking application where front end, back end, networking etc is included as a single package. Suppose developer decides to add some new feature to the front end, it might end up affecting whole application. Also it takes lot of time for a change to go into production(may be months) and get feedback because we never know what will happen if the feature goes into production.

To get rid of this problem faced in waterfall model, agile model came into existence, which is based on philosophy to rapidly deploy application without waiting and ship small chunks of code to client.

Take example of amazon.com. It is not a single application, it is composed of multiple services like front end, catalog, cart and payment. If developer is working on one service, he need not worry about other service and developer gets feedback before the code goes into production and since each service is separate, changes made to one service does not affect other.

Actors: Product owner, Scrum master and Team(Dev,Test).

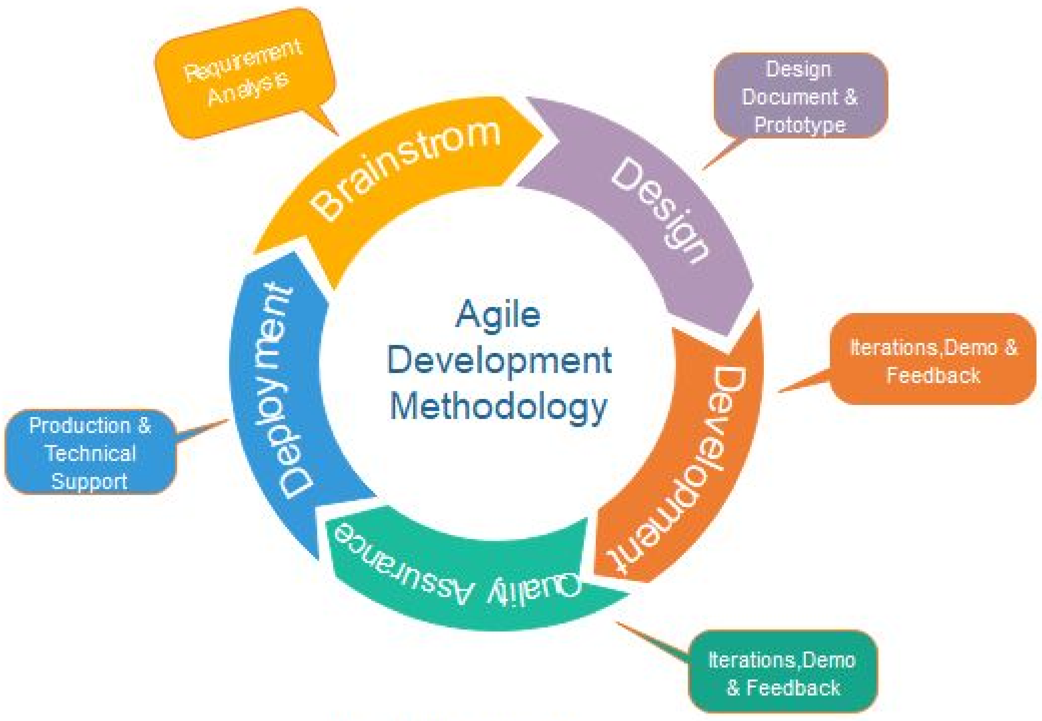

Requirement Analysis
Brainstrom
Design Document & Prototype
Design
Iterations,Demo & Feedback
Development
Iterations,Demo & Feedback
Quality Assurance
Deployment
Production & Technical Support
Agile Development Methodology

Fig. Agile Model

Phases of Agile Model:

Following are the phases in the Agile model are as follows:

1. Requirements gathering
2. Design the requirements
3. Construction/ iteration
4. Testing/ Quality assurance
5. Deployment
6. Feedback

1. Requirements gathering: In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility.

2. Design the requirements: When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.

3. Construction/ iteration: When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.

4. Testing: In this phase, the Quality Assurance team examines the product's performance and looks for the bug.

5. Deployment: In this phase, the team issues a product for the user's work environment.

6. Feedback: After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feedback.

Agile Testing Methods:

- Scrum
- Crystal
- Dynamic Software Development Method(DSDM)
- Feature Driven Development(FDD)
- Lean Software Development
- eXtreme Programming(XP)

Scrum

SCRUM is an agile development process focused primarily on ways to manage tasks in team-based development conditions.

There are three roles in it, and their responsibilities are:

- Scrum Master: The scrum can set up the master team, arrange the meeting and remove obstacles for the process
- Product owner: The product owner makes the product backlog, prioritizes the delay and is responsible for the distribution of functionality on each repetition.
- Scrum Team: The team manages its work and organizes the work to complete the sprint or cycle.

eXtreme Programming(XP)

This type of methodology is used when customers are constantly changing demands or requirements, or when they are not sure about the system's performance.

Crystal:

There are three concepts of this method-

1. Chartering: Multi activities are involved in this phase such as making a development team, performing feasibility analysis, developing plans, etc.
2. Cyclic delivery: under this, two more cycles consist, these are:
    - Team updates the release plan.
    - Integrated product delivers to the users.
3. Wrap up: According to the user environment, this phase performs deployment, post-deployment.

Dynamic Software Development Method(DSDM):

DSDM is a rapid application development strategy for software development and gives an agile project distribution structure. The essential features of DSDM are that users must be actively connected, and teams have been given the right to make decisions. The techniques used in DSDM are:

1. Time Boxing
2. MoSCoW Rules
3. Prototyping

The DSDM project contains seven stages:

1. Pre-project
2. Feasibility Study
3. Business Study
4. Functional Model Iteration
5. Design and build Iteration
6. Implementation
7. Post-project

Feature Driven Development(FDD):

This method focuses on "Designing and Building" features. In contrast to other smart methods, FDD describes the small steps of the work that should be obtained separately per function.

Lean Software Development:

Lean software development methodology follows the principle "just in time production." The lean method indicates the increasing speed of software development and reducing costs. Lean development can be summarized in seven phases.

1. Eliminating Waste
2. Amplifying learning
3. Defer commitment (deciding as late as possible)
4. Early delivery
5. Empowering the team
6. Building Integrity
7. Optimize the whole

When to use the Agile Model?

- When frequent changes are required.
- When a highly qualified and experienced team is available.
- When a customer is ready to have a meeting with a software team all the time.
- When project size is small.

Advantage(Pros) of Agile Method:

1. Frequent Delivery
2. Face-to-Face Communication with clients.

3. Efficient design and fulfils the business requirement.

4. Anytime changes are acceptable.

5. It reduces total development time.

Disadvantages(Cons) of Agile Model:

1. Due to the shortage of formal documents, it creates confusion and crucial decisions taken throughout various phases can be misinterpreted at any time by different team members.

2. Due to the lack of proper documentation, once the project completes and the developers allotted to another project, maintenance of the finished project can become a difficulty.

7. Iterative Model

In this Model, you can start with some of the software specifications and develop the first version of the software. After the first version if there is a need to change the software, then a new version of the software is created with a new iteration. Every release of the Iterative Model finishes in an exact and fixed period that is called iteration.

The Iterative Model allows the accessing earlier phases, in which the variations made respectively. The final output of the project renewed at the end of the Software Development Life Cycle (SDLC) process.

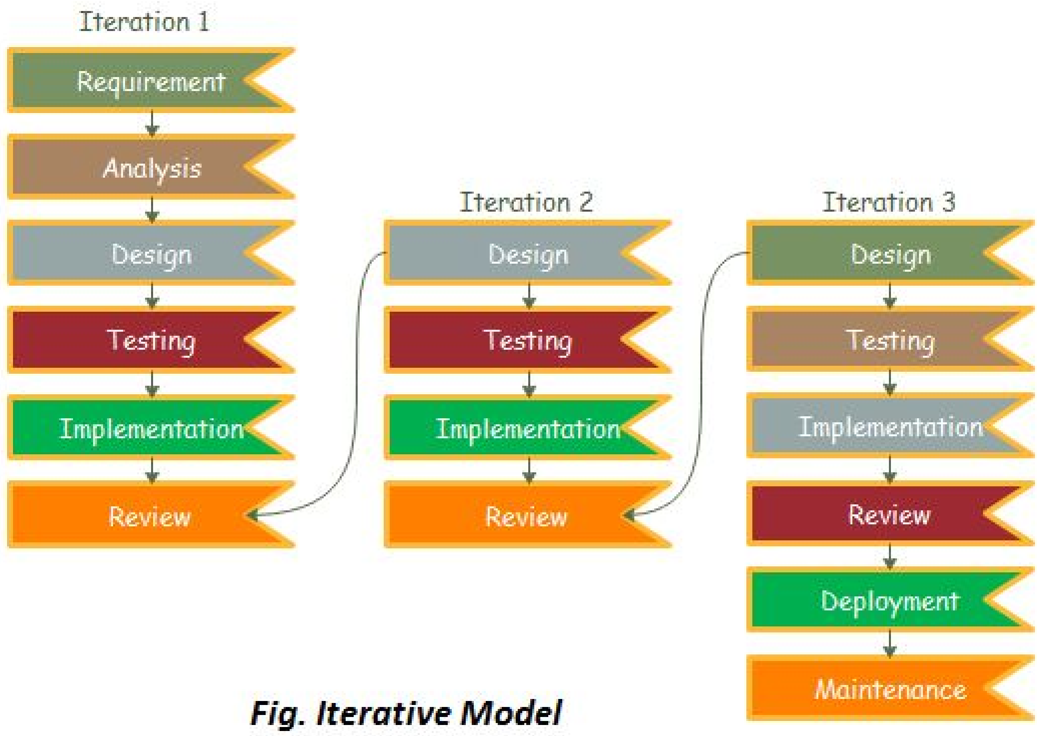


Fig. Iterative Model

The various phases of Iterative model are as follows:

1. Requirement gathering & analysis: In this phase, requirements are gathered from customers and check by an analyst whether requirements will fulfil or not. Analyst checks that need will achieve within budget or not. After all of this, the software team skips to the next phase.

2. Design: In the design phase, team design the software by the different diagrams like Data Flow diagram, activity diagram, class diagram, state transition diagram, etc.

3. Implementation: In the implementation, requirements are written in the coding language and transformed into computer programmes which are called Software.

4. Testing: After completing the coding phase, software testing starts using different test methods. There are many test methods, but the most common are white box, black box, and grey box test methods.

5. Deployment: After completing all the phases, software is deployed to its work environment.

6. Review: In this phase, after the product deployment, review phase is performed to check the behaviour and validity of the developed product. And if there are any error found then the process starts again from the requirement gathering.

7. Maintenance: In the maintenance phase, after deployment of the software in the working environment there may be some bugs, some errors or new updates are required. Maintenance involves debugging and new addition options.

When to use the Iterative Model?

1. When requirements are defined clearly and easy to understand.
2. When the software application is large.
3. When there is a requirement of changes in future.

Advantage(Pros) of Iterative Model:

1. Testing and debugging during smaller iteration is easy.
2. A Parallel development can plan.
3. It is easily acceptable to ever-changing needs of the project.
4. Risks are identified and resolved during iteration.
5. Limited time spent on documentation and extra time on designing.

Disadvantage(Cons) of Iterative Model:

1. It is not suitable for smaller projects.
2. More Resources may be required.
3. Design can be changed again and again because of imperfect requirements.
4. Requirement changes can cause over budget.
5. Project completion date not confirmed because of changing requirements.

8. Big Bang Model

In this model, developers do not follow any specific process. Development begins with the necessary funds and efforts in the form of inputs. And the result may or may not be as per the customer's requirement, because in this model, even the customer requirements are not defined.

This model is ideal for small projects like academic projects or practical projects. One or two developers can work together on this model.

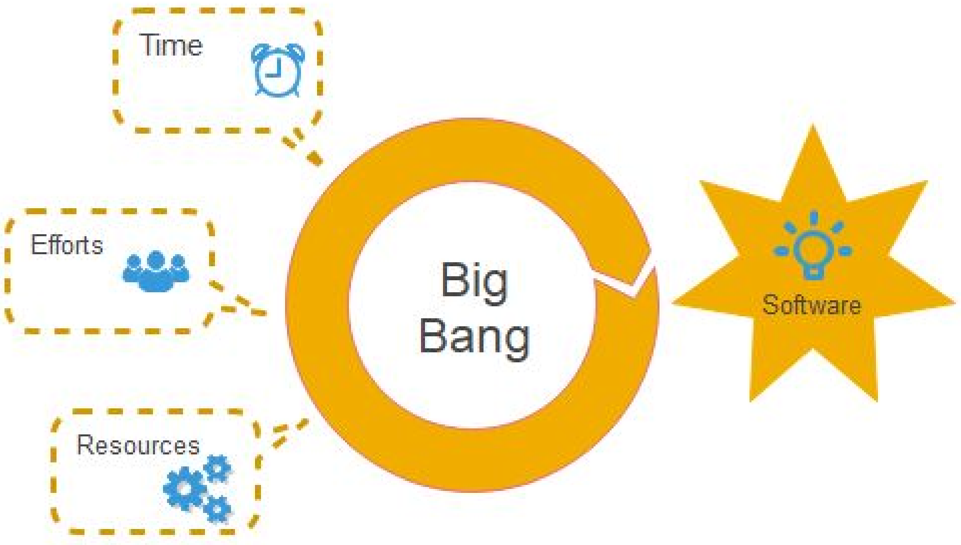


Fig. Big Bang Model

When to use Big Bang Model?

As we discussed above, this model is required when this project is small like an academic project or a practical project. This method is also used when the size of the developer team is small and when requirements are not defined, and the release date is not confirmed or given by the customer.

Advantage(Pros) of Big Bang Model:

1. There is no planning required.
2. Simple Model.
3. Few resources required.
4. Easy to manage.
5. Flexible for developers.

Disadvantage(Cons) of Big Bang Model:

1. There are high risk and uncertainty.
2. Not acceptable for a large project.
3. If requirements are not clear that can cause very expensive.

<u>**TestingObjectives:**</u>

Verification

Validation

Find Defects

Prevent Defects

Providing Information

Improve quality of software

Verifying performance & functionality

For optimum user experience

<u>**Levels of Testing**</u>

There are four levels of testing:

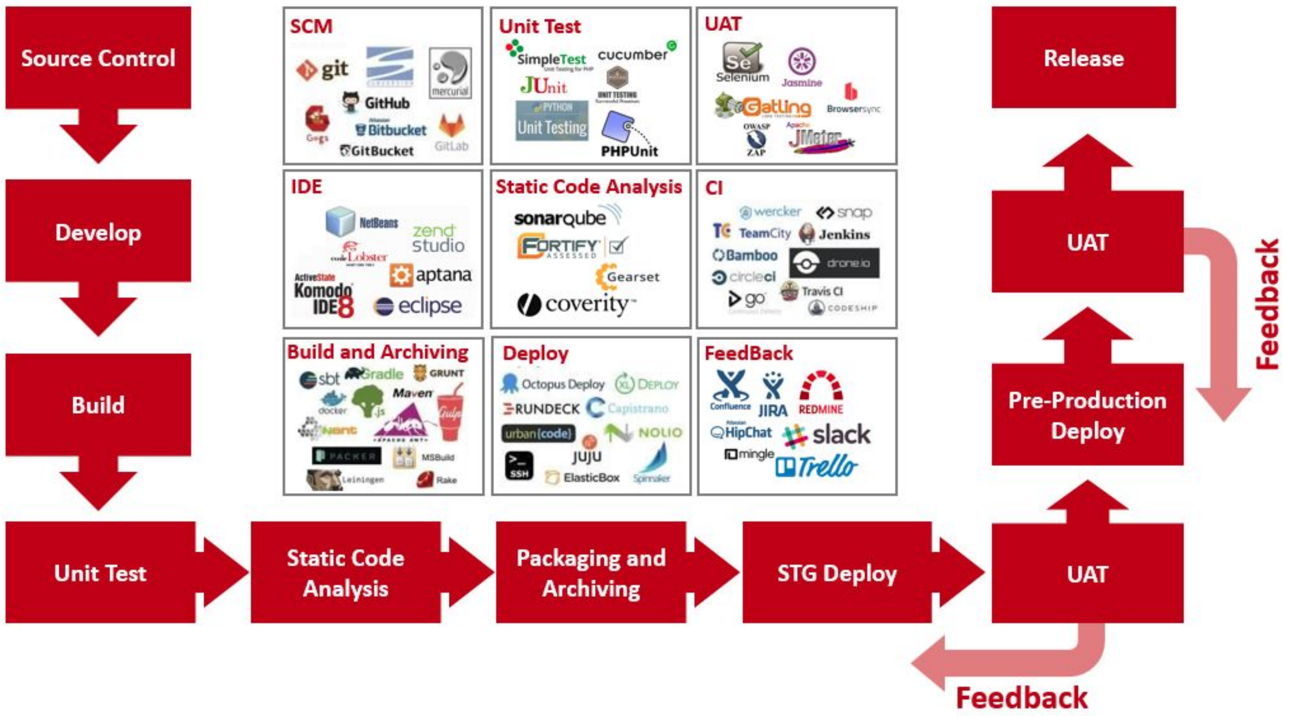
unit testing

integration testing

system testing

acceptance testing

CI/CD Tools:



Code Quality:

Clean code makes you work fast

Measure the size of functions

Name your functions well

Build a Code Quality Assurance System for your team

Version control tool to ensure code quality and transparency (e.gGit)

Style guide for readable and comprehensible code•

Use linters to automatically test code style (C C++ PC-lint, Python pylint, JavaScript ESLint)

Improve code quality with functional tests (unit, integration, system)

Track Test Coverage (e.gLDRA)

1.Statement coverage(%): number of statements executed during a test divided by all statements

2.Branch coverage(%): number of executed conditions divided by all conditions

3.Function coverage(%): number of executed functions divided by all functions

4.Lines coverage(%): number of lines ran during a test divided by all linesUse Continuous Integration Tools (e.gJenkis, TravisCI)

Difference between Agile Model and V-Model

Agile Model:

Agile Model is the software development model in which the development and testing process carries on simultaneously. In this model, both development related processes and testing related processes are parallel. This model provides the facility of more interaction between development team, testing team and end-users.

V-Model:

V-Model is the software development model in which testing takes place once the development process is fully complete or almost complete. In V-Model development and testing processes are kept quite separate. It is not as reliable as the Agile Model.

Difference between Agile Model and V-Model:

AGILE MODEL	V-MODEL
It is a software development model in which development and testing are concurrent.	It is also a software development model but development and testing are not concurrent.

It consists of different sprints.	It has two phases Verification and Validation.
Testing is easy as compared to V-model.	Testing is hard as compared to the Agile model.
It consists of a total of five phases.	It consists of five verification and five validation phases.
Communication is easy between end users, development team and testing team.	Development team and testing team don’t interact much with end users.
Developers and testers are dependent on each other.	Developers and testers are independent.
It is iterative.	It is not iterative.
It is incremental.	It is not incremental.

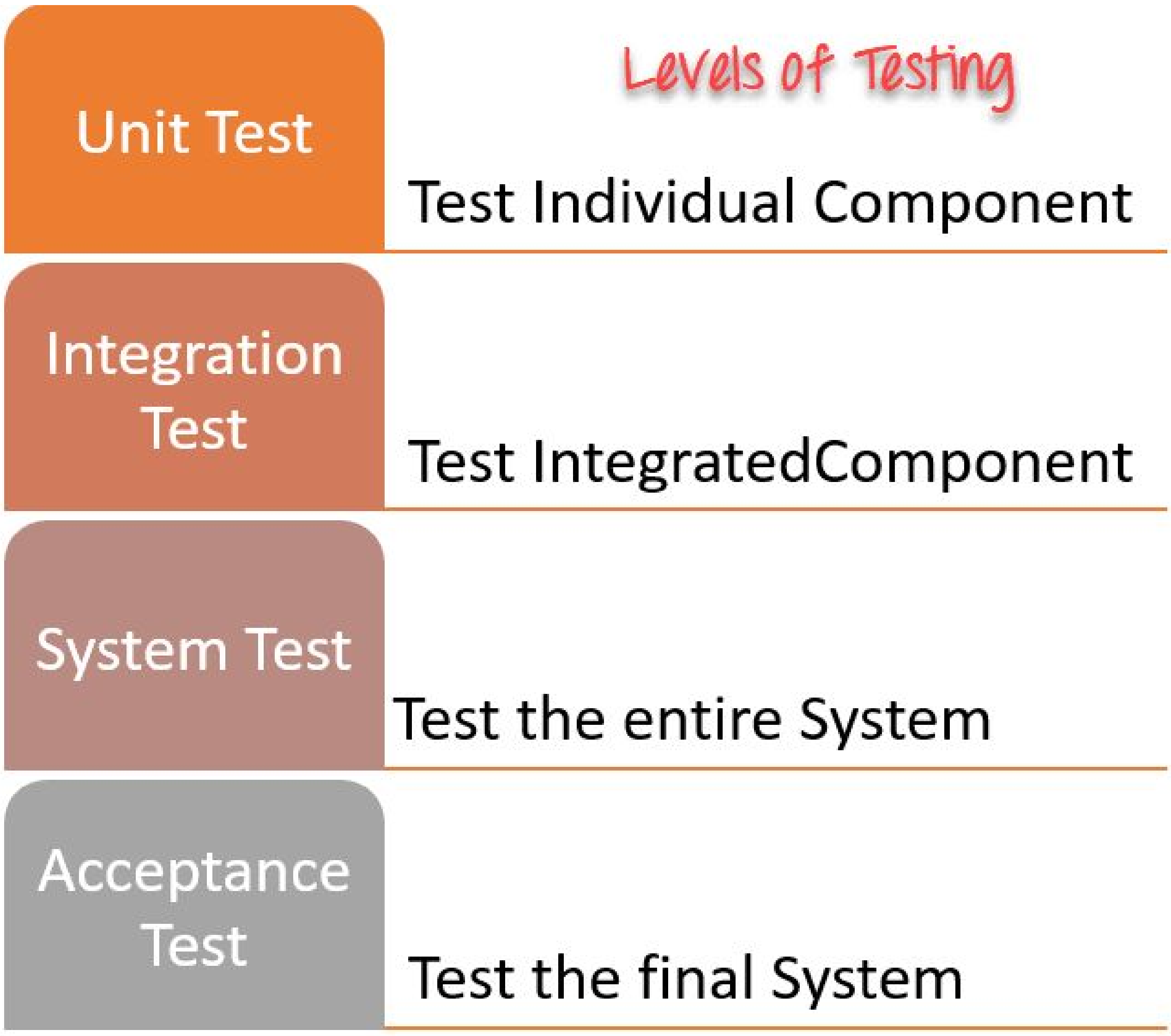
TESTING PROCESS

https://www.simplilearn.com/fundamentals-of-software-testing-tutorial-video

Levels of Testing

There are mainly four **Levels of Testing** in software testing :

1. **Unit Testing** : checks if software components are fulfilling functionalities or not.
2. **Integration Testing** : checks the data flow from one module to other modules.
3. **System Testing** : evaluates both functional and non-functional needs for the testing.
4. **Acceptance Testing** : checks the requirements of a specification or contract are met as per its delivery.



Each of these testing levels has a specific purpose. These testing level provide value to the software development lifecycle.

1) **Unit testing:**

A Unit is a smallest testable portion of system or application which can be compiled, liked, loaded, and executed. This kind of testing helps to test each module separately.

The aim is to test each part of the software by separating it. It checks that component are fulfilling functionalities or not. This kind of testing is performed by developers.

2) **Integration testing:**

Integration means combining. For Example, In this testing phase, different software modules are combined and tested as a group to make sure that the integrated system is ready for system testing.

Integrating testing checks the data flow from one module to other modules. This kind of testing is performed by testers.

3) **System testing:**

System testing is performed on a complete, integrated system. It allows checking the system's compliance as per the requirements. It tests the overall interaction of components. It involves load, performance, reliability and security testing.

System testing most often the final test to verify that the system meets the specification. It evaluates both functional and non-functional need for the testing.

4) **Acceptance testing:**

Acceptance testing is a test conducted to find if the requirements of a specification or contract are met as per its delivery. Acceptance testing is basically done by the user or customer. However, other stockholders can be involved in this process.\

Testing methodologies

Functional vs. Non-functional Testing

The goal of utilizing numerous testing methodologies in your development process is to make sure your software can successfully operate in multiple environments and across different platforms. These testing methods are usually conducted in order and include:

- Unit testing
- Integration testing
- System testing
- Acceptance testing

Non-functional testing methods incorporate all test types focused on the operational aspects of a piece of software. These include:

- Performance testing
- Security testing
- Usability testing
- Compatibility testing

Unit Testing

Unit testing is the first level of testing and is often performed by the developers themselves. It is the process of ensuring individual components of a piece of software at the code level are functional and work as they were designed to.

Unit testing can be conducted manually, but automating the process will speed up delivery cycles and expand test coverage.

Integration Testing

These tests are often framed by user scenarios, such as logging into an application or opening files. Integrated tests can be conducted by either developers or independent testers and are usually comprised of a combination of automated functional and manual tests.

System Testing

System testing is a black box testing method used to evaluate the completed and integrated system, as a whole, to ensure it meets specified requirements.

The functionality of the software is tested from end-to-end and is typically conducted by a separate testing team than the development team before the product is pushed into production.

Acceptance Testing

Acceptance testing is the last phase of functional testing and is used to assess whether or not the final piece of software is ready for delivery.

This requires the product be tested both internally and externally

Beta testing is key to getting real feedback from potential customers and can address any final usability concerns.

Performance Testing

Performance testing is a non-functional testing technique used to determine how an application will behave under various conditions. Performance testing can be broken down into four types:

- **Load testing** is the process of putting increasing amounts of simulated demand on your software, application, or website to verify whether or not it can handle what it’s designed to handle.
- **Stress testing** takes this a step further and is used to gauge how your software will respond at or beyond its peak load.
- **Endurance testing,** also known as soak testing, is used to analyze the behavior of an application under a specific amount of simulated load over longer amounts of time.
- **Spike testing** is a type of load test used to determine how your software will respond to substantially larger bursts of concurrent user or system activity over varying amounts of time.

Security Testing

Security testing is a non-functional software testing technique used to determine if the information and data in a system is protected. The goal is to purposefully find loopholes and security risks in the system that could result in unauthorized access to or the loss of information by probing the application for weaknesses.

There are multiple types of this testing method, each of which aimed at verifying six basic principles of security:

1. Integrity
2. Confidentiality
3. Authentication
4. Authorization
5. Availability
6. Non-repudiation

Usability Testing

Usability testing is a testing method that measures an application's ease-of-use from the end-user perspective and is often performed during the system or acceptance testing stages.

Compatibility Testing

Compatibility testing is used to gauge how an application or piece of software will work in different environments. It is used to check that your product is compatible with multiple operating systems, platforms, browsers, or resolution configurations.