**Major GIT commands:**

* **git config**

**Sets configuration values for your user name, email, gpg key, preferred diff algorithm, file formats and more. Example: git config --global user.name "My Name" git config --global user.email "user@domain.com" cat ~/.gitconfig [user] name = My Name email = user@domain.com**

* **git init**

**Initializes a git repository – creates the initial ‘.git’ directory in a new or in an existing project. Example: cd /home/user/my\_new\_git\_folder/ git init**

* **git clone**

**Makes a Git repository copy from a remote source. Also adds the original location as a remote so you can fetch from it again and push to it if you have permissions. Example: git clone git@github.com:user/test.git**

* **git add**

**Adds files changes in your working directory to your index. Example: git add .**

* **git rm**

**Removes files from your index and your working directory so they will not be tracked. Example: git rm filename**

* **git commit**

**Takes all of the changes written in the index, creates a new commit object pointing to it and sets the branch to point to that new commit. Examples: git commit -m ‘committing added changes’ git commit -a -m ‘committing all changes, equals to git add and git commit’**

* **git status**

**Shows you the status of files in the index versus the working directory. It will list out files that are untracked (only in your working directory), modified (tracked but not yet updated in your index), and staged (added to your index and ready for committing). Example: git status # On branch master # # Initial commit # # Untracked files: # (use "git add <file>..." to include in what will be committed) # # README nothing added to commit but untracked files present (use "git add" to track)**

* **git branch**

**Lists existing branches, including remote branches if ‘-a’ is provided. Creates a new branch if a branch name is provided. Example: git branch -a \* master remotes/origin/master**

* **git checkout**

**Checks out a different branch – switches branches by updating the index, working tree, and HEAD to reflect the chosen branch. Example: git checkout newbranch**

* **git merge**

**Merges one or more branches into your current branch and automatically creates a new commit if there are no conflicts. Example: git merge newbranchversion**

* **git reset**

**Resets your index and working directory to the state of your last commit. Example: git reset --hard HEAD**

* **git stash**

**Temporarily saves changes that you don’t want to commit immediately. You can apply the changes later. Example: git stash Saved working directory and index state "WIP on master: 84f241e first commit" HEAD is now at 84f241e first commit (To restore them type "git stash apply")**

* **git tag**

**Tags a specific commit with a simple, human readable handle that never moves. Example: git tag -a v1.0 -m 'this is version 1.0 tag'**

* **git fetch**

**Fetches all the objects from the remote repository that are not present in the local one. Example: git fetch origin**

* **git pull**

**Fetches the files from the remote repository and merges it with your local one. This command is equal to the git fetch and the git merge sequence. Example: git pull origin**

* **git push**

**Pushes all the modified local objects to the remote repository and advances its branches. Example: git push origin master**

* **git remote**

**Shows all the remote versions of your repository. Example: git remote origin**

* **git log**

**Shows a listing of commits on a branch including the corresponding details. Example: git log commit 84f241e8a0d768fb37ff7ad40e294b61a99a0abe Author: User <user@domain.com> Date: Mon May 3 09:24:05 2010 +0300 first commit**

* **git show**

**Shows information about a git object. Example: git show commit 84f241e8a0d768fb37ff7ad40e294b61a99a0abe Author: User <user@domain.com> Date: Mon May 3 09:24:05 2010 +0300 first commit diff --git a/README b/README new file mode 100644 index 0000000..e69de29**

* **git ls-tree**

**Shows a tree object, including the mode and the name of each item and the SHA-1 value of the blob or the tree that it points to. Example: git ls-tree master^{tree} 100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391 README**

* **git cat-file**

**Used to view the type of an object through the SHA-1 value. Example: git cat-file -t e69de29bb2d1d6434b8b29ae775ad8c2e48c5391 blob**

* **git grep**

**Lets you search through your trees of content for words and phrases. Example: git grep "www.siteground.com" -- \*.php**

* **git diff**

**Generates patch files or statistics of differences between paths or files in your git repository, or your index or your working directory. Example: git diff**

* **gitk**

**Graphical Tcl/Tk based interface to a local Git repository. Example: gitk**

* **git instaweb**

**Runs a web server with an interface into your local repository and automatically directs a web browser to it. Example: git instaweb --httpd=webrick git instaweb --stop**

* **git archive**

**Creates a tar or zip file including the contents of a single tree from your repository. Example: git archive --format=zip master^ README >file.zip**

* **git gc**

**Garbage collector for your repository. Optimizes your repository. Should be run occasionally. Example: git gc Counting objects: 7, done. Delta compression using up to 2 threads. Compressing objects: 100% (5/5), done. Writing objects: 100% (7/7), done. Total 7 (delta 1), reused 0 (delta 0)**

* **git fsck**

**Does an integrity check of the Git file system, identifying corrupted objects. Example: git fsck**

* **git prune**

**Removes objects that are no longer pointed to by any object in any reachable branch. Example: git prune**

**Interview question**

**1) Explain what is JIRA?**

JIRA is an issue tracking product or a software tool developed by Atlassian, commonly used for bug tracking, project management and issue tracking; it is entirely based on this three aspects.

**2) Explain what is a workflow?**

Workflow is defined as a movement of the bug/issue through various stages during its life-cycle

Created/Open

WIP ( Work In Progress)

Completed/Closed

**3) What can be referred as an issue in JIRA?**

In JIRA, an issue can be anything like a

* Software bug
* The project task
* A help-desk ticket
* The leave request form

**4) List out the source control programs with which it integrates?**

It integrates with source control programs such as CVS, Git, Subversion, Clearcase, Visual SourceSafe, Mercurial, and Perforce.

**5) Why use JIRA?**

The reason behind using JIRA is

* Upfront and fair licensing policy
* Features that is not available elsewhere
* Get latest update on the progress of projects
* It run anywhere and recognized with many famous companies
* Easily extensible and customizable

**6) Is it possible to access JIRA cloud site via a mobile device?**

Yes, it is possible to access JIRA cloud site via a mobile device. You have to just use the URL of the JIRA cloud site in your mobile web browser.

**7) Can you disable JIRA mobile for the site?**

You can disable JIRA mobile for the site, so that users can be unable to operate the desktop view of JIRA on their mobile device.  JIRA mobile comes as a system add-on and can be disabled any time.

**8) Explain labelling and linking issue in JIRA?**

* **Labelling Issue:** It enables you to categorize an issue in a more informal way than assigning it to a component or version. You can then search issues according to label.
* **Linking** **Issue:** This feature enables you to link an association between two issues on either on the same or different JIRA servers.

**9) Mention the types of reports generated in JIRA?**

JIRA offer reports that show statistics for projects, versions, people or other fields within issues.  Various reports included with JIRA are

* Average Age Report
* Pie Chart Report
* Resolution Time Report
* Recently Created Issues Report
* Resolved vs. Created Issues Report
* Single Level Group by Report
* Time Tracking Report
* User Workload Report
* Workload Pie Chart Report, etc.

**10) Explain what is Cloning an Issue?**

Cloning as issue allows you to create a duplicate of the original issue so that many employees can work on a single issue within a single project. The clone issue can be connected to the original issue.  A clone issue holds following the information

* Summary
* Description
* Assignee
* Environment
* Priority
* Issue Type
* Security
* Reporter
* Components, etc.

**11) Mention what things are not included in cloned issue in JIRA?**

* Time tracking
* Issue history
* Comments

**12) Explain what is the use of “Move Issue” wizard in JIRA?**

The move issue wizard enables you to specify another project in your JIRA instance. Move wizard permit you to change certain attributes of an issue like

* **Issue Type:** If your issue is a custom issue type and does not occur in your target project, you must choose a new issue type for your issue
* **Issue Status:** If you have assigned your issue as a custom issue status and it does not exist in your project, you must select a new issue status for your issue
* **Custom Fields:** If you have determined required custom fields for your issue, which do not occur in the target project, you must set values for them.

**13) How security setting is helpful in JIRA?**

JIRA’S security setting restricts the access to the issue to only those person who is allowed to work on the issue or a member of the chosen security level. Security level of an issue can be set either when the issue is created or when the issue is being edited

**14) Explain how you can share an issue with other users?**  
You can email an issue by using the share option in JIRA. You can also email other JIRA users a link to the issue by sharing the issue with them or by mentioning them in an issue’s Description or Comment field.

**15) Explain how you can modify multiple bulk issues?**

To modify multiple bulk issues, you can use **Bulk Change** option from the “Tools” menu of the navigator.  All the issues on the current page can be selected for the bulk operation.  The following list details the available bulk operations like

* Workflow Transition
* Delete
* Move
* Edit

**16) Explain how you can disable mail notification for Bulk Operations?**

To disable mail notification for a particular Bulk Operations, you have to de-select the “Send Notification” checkbox in the bulk operation wizard.

**17) What does an issue change history include?**

Issue change history includes

* Deletion of a comment
* Deletion of a worklog
* Creation or deletion of an issue link
* Attachment of a file
* Changes to an issue field

**18) Explain what does the three color indicates tracking times or duration for an issue?**

Three color will be displayed representing the amount of time spent behind the issue

* Original Estimate (Blue): The amount of time originally estimated to resolve the issue
* Remaining Estimate(Orange): The remaining amount of time left to resolve the issue
* Time Spen or Logged (Green): The amount of time spent so far while resolving the issue

19 Diffenerence between bugzila n jira

Bugzilla is defect tracking tool, its doing what is supposed to do: track bugs, send email notification....etc  
The good future I like in bugzilla is that you can easily find similar defect when entering new defect in the system.  
Defect title field will list all similar defects by keywords you are entering..... and I think bugzilla is free.  
  
Jira is not free, but its very powerful. Its for agile project and might require other sister tools from <https://www.atlassian.com/> that you might need, such as Confluence, JiraAgile(previously greenhoper), fisheye...etc.  
  
With Jira only, you can track stories and defects, create filters and plan your work.  
I recently got a add-on for my team called Zephyr for Jira, which is a great test cases management and test cycle planing tool. But again these addons cost money.

**epic **

An epic captures a large body of work. It is essentially a large [user story](https://confluence.atlassian.com/display/AGILE/Story) that can be broken down into a number of smaller stories. It may take several [sprints](https://confluence.atlassian.com/display/AGILE/Sprint) to complete an epic.

In JIRA Agile, an epic is simply an issue of type "Epic", which has a name (e.g. "My Epic") with which other issues can be associated (linked).

* This page only applies to [Scrum boards](https://confluence.atlassian.com/display/AGILE/Scrum+Board).

***STORY***

*story* or *user story* is a software system requirement that is expressed in a few short sentences, ideally using non-technical language.

* In JIRA Agile, a story is represented as an [issue](https://confluence.atlassian.com/display/AGILE/Creating+an+Issue), and individual tasks within the story are represented as [sub-tasks](https://confluence.atlassian.com/display/AGILE/Creating+a+Sub-Task). To see an example of a board showing stories in a backlog and in a sprint, see [Using Plan Mode](https://confluence.atlassian.com/display/AGILE/Using+Plan+Mode).

**sprint **

* A sprint — also known as an iteration — is a short (ideally two to four week) period in which the development team implements and delivers a discrete product increment, e.g. a working milestone version.
* In JIRA Agile, go to [Plan mode](https://confluence.atlassian.com/display/AGILE/Using+Plan+Mode) on your board to [create a sprint](https://confluence.atlassian.com/display/AGILE/Creating+a+Sprint), choose issues and [start a sprint](https://confluence.atlassian.com/display/AGILE/Starting+a+Sprint). Then go to [Work mode](https://confluence.atlassian.com/display/AGILE/Using+Work+Mode) to work on issues and [complete the sprint](https://confluence.atlassian.com/display/AGILE/Ending+a+Sprint). For completed sprints, you can view the [Sprint Retrospective Report](https://confluence.atlassian.com/display/AGILE/Viewing+the+Sprint+Report) in Report mode.