**Clone**

Clone operation creates the instance of the repository. Clone operation not only checks out the working copy, but it also mirrors the complete repository. Users can perform many operations with this local repository.

### Pull

Pull operation copies the changes from a remote repository instance to a local one.

### Push

Push operation copies changes from a local repository instance to a remote one.

### HEAD

### HEAD is a pointer, which always points to the latest commit in the branch.

The heads of the branches are stored in **.git/refs/heads/** directory.

**Cherry Pick**

**Cherry picking** in **Git** is designed to apply some commit from one branch into another branch. It can be done if you eg. made a mistake and committed a change into wrong branch, but do not want to merge the whole branch. You can just eg. revert the commit and **cherry**-**pick** it on another branch.

The **cherry-pick** command "copies" a commit, creating a new commit on the current branch with the same message and patch as another commit.

Required changes will be copy with the help of **SHA1**

**Rebase**

A rebase is an alternative to a [**merge**](https://marklodato.github.io/visual-git-guide/index-en.html#merge) for combining multiple branches. Whereas a merge creates a single commit with two parents, leaving a non-linear history, a rebase replays the commits from the current branch onto another, leaving a linear history. In essence, this is an automated way of performing several [**cherry-pick**](https://marklodato.github.io/visual-git-guide/index-en.html#cherry-pick)**s** in a row.

**Merge**

A **merge** creates a new commit that incorporates changes from other commits. Before merging, the stage must match the current commit.

**Reset**

The **reset** command moves the current branch to another position, and optionally updates the stage and the working directory. It also is used to copy files from the history to the stage without touching the working directory.

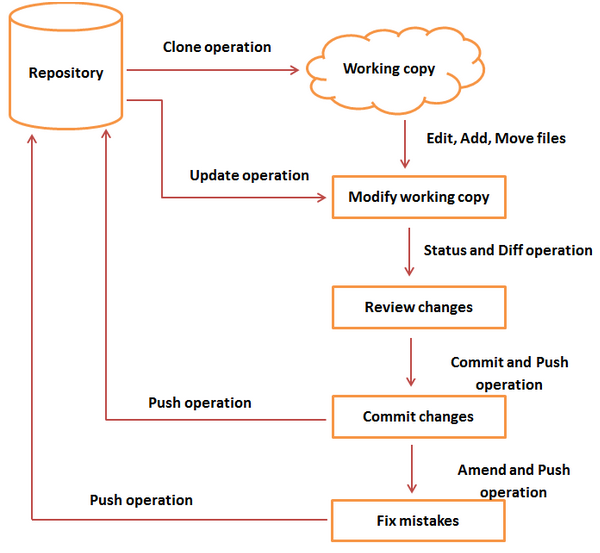
### Checkout

The **checkout** command is used to copy files from the history (or stage) to the working directory, and to optionally switch branches.

Checkout will switch the branch, and reset the files as previous.

### Diff

There are various ways to look at differences between commits. Below are some common examples. Any of these commands can optionally take extra filename arguments that limit the differences to the named files.



1. How do you find a list of files that has changed in a particular commit?

git diff-tree --no-commit-id --name-only -r a05828b3d4ef36352ec75c6d168b164aea365978

1. How do you revert a commit that has already been pushed and made public?

git revert SHA1 [Which you reverted]

1. How do you squash last N commits into a single commit?

git rebase -i SHA1

1. How do you setup a script to run every time a repository receives new commits through push?

To configure a script to run every time a repository receives new commits through push, one needs to define either a pre-receive, update, or a post-receive hook depending on when exactly the script needs to be triggered.

Pre-receive hook in the destination repository is invoked when commits are pushed to it. Any script bound to this hook will be executed before any references are updated. This is a useful hook to run scripts that help enforce development policies.

1. What is git bisect? How can you use it to determine the source of a (regression) bug?

* git-bisect - Use binary search to find the commit that introduced a bug
* git bisect start, the repository enters bisect mode. After this, all you have to do is identify a bad and a good commit:

$ git bisect <subcommand> <options>

$ git bisect bad # marks the current version as bad

$ git bisect good {hash or tag}

# marks the given hash or tag as good, ideally of some earlier commit

1. What are the different ways you can refer to a commit?
2. What is git rebase and how can it be used to resolve conflicts in a feature branch before merge?

In simple words, git rebase allows one to move the first commit of a branch to a new starting location. For example, if a feature branch was created from master, and since then the master branch has received new commits, git rebase can be used to move the feature branch to the tip of master. The command effectively will replay the changes made in the feature branch at the tip of master, allowing conflicts to be resolved in the process. When done with care, this will allow the feature branch to be merged into master with relative case and sometimes as a simple fast-forward operation.

1. How do you configure a Git repository to run code sanity checking tools right before making commits, and preventing them if the test fails?

This can be done with a simple script bound to the pre-commit hook of the repository. The pre-commit hook is triggered right before a commit is made, even before you are required to enter a commit message. In this script one can run other tools, such as linters and perform sanity checks on the changes being committed into the repository. For example, the following script:

#!/bin/sh

files=$(git diff --cached --name-only --diff-filter=ACM | grep '.go$')

if [ -z files ]; then

exit 0

fi

unfmtd=$(gofmt -l $files)

if [ -z unfmtd ]; then

exit 0

fi

echo “Some .go files are not fmt’d”

exit 1

… checks to see if any .go file that is about to be commited needs to be passed through the standard Go source code formatting tool gofmt. By exiting with a non-zero status, the script effectively prevents the commit from being applied to the repository.

1. Git commands

$ git config --global user.name "Sam Smith"

$ git config --global user.email [rahamv9@example.com](mailto:rahamv9@example.com)

[Create a new local repository](http://atlassian.com/git/tutorial/git-basics#%21init)**: $** git init repo\_name

1. Create a working copy of a local repository:

$ git clone /path/to/repository git-name

$ git clone rahamv9@host:/path/to/repository

1. Add one or more files to staging (index):

$ git add file name/ git add \*/git add .

1. Commit changes to head (but not yet to the remote repository):

$ git commit -m "Commit message"

1. Commit any files you've added with git add, and also commit any files you've changed since then:

$ git commit -a

1. Send changes to the master branch of your remote repository:

$ git push origin master

$ git push origin HEAD:refs/for/Branch Name/Topic Name

1. List the files you've changed and those you still need to add or commit

$ git status

1. If you haven't connected your local repository to a remote server, add the server to be able to push to it:

$ git remote add origin <server>

1. List all currently configured remote repositories:

$ git remote -v

1. Create a new branch and switch to it:

$ git checkout -b <branchname>

1. Switch from one branch to another:

$ git checkout <branchname>

1. List all the branches in your repo, and also tell you what branch you're currently in:

$ git branch

1. Delete the feature branch:

$ git branch -d <branchname>

1. Push the branch to your remote repository, so others can use it:

$ git push origin <branchname>

1. Push all branches to your remote repository:

$ git push --all origin

1. Delete a branch on your remote repository:

$ git push origin :<branchname>

1. Fetch and merge changes on the remote server to your working directory:

$ git pull

1. To merge a different branch into your active branch:

$ git merge <branchname>

$ git merge origin/n-tone

1. View all the merge conflicts:

$ git diff

1. View the conflicts against the base file:

$ git diff --base <filename>

1. Preview changes, before merging

$ git diff <sourcebranch> <targetbranch>

1. After you have manually resolved any conflicts, you mark the changed file:

$ git add <filename>

1. You can use tagging to mark a significant change set, such as a release:

$ git tag 1.0.0 <commitID>

1. Commit ID is the leading characters of the change set ID, up to 10, but must be unique. Get the ID using:

$ git log

1. Push all tags to remote repository:

$ git push --tags origin

1. If you mess up, you can replace the changes in your working tree with the last content in head:

Changes already added to the index, as well as new files, will be kept.

$ git checkout -- <filename>

1. Search the working directory for foo():

$ git grep "any string"

1. Deference between git reset and git revet

Git revert: collecting back needed changes

Git reset: On the commit-level, resetting is a way to move the tip of a branch to a different commit. This can be used to remove commits from the current branch.

|  |  |  |
| --- | --- | --- |
| git | reset | Commit-level Discard commits in a private branch or throw away uncommitted changes |
| git | reset | File-level Unstage a file |
| git | checkout | Commit-level Switch between branches or inspect old snapshots |
| git | checkout | File-level Discard changes in the working directory |
| git | revert | Commit-level Undo commits in a public branch |

1. We’ve broken down the SVN-to-Git migration process into 5 simple steps:

* Prepare your environment for the migration.
* Convert the SVN repository to a local Git repository.
* Synchronize the local Git repository when the SVN repository changes.
* Share the Git repository with your developers via Bitbucket.
* Migrate your development efforts from SVN to Git.

**Prepare**, **convert**, and **synchronize** steps take a **SVN** commit history and turn it into a **Git** repository. The best way to manage these first 3 steps is to designate one of your team members as the migration lead (if you’re reading this guide, that person is probably you). All 3 of these steps should be performed on the migration lead’s local computer.

After the synchronize phase, the migration lead should have no trouble keeping a local Git repository up-to-date with an SVN counterpart. To share the Git repository, the migration lead can share his local Git repository with other developers by pushing it to [Bitbucket](http://bitbucket.org), a Git hosting service.