GITHUB BASED GIT USER MANUAL

BY

insa gEOSPATIAL dIRECTORATE r&d tEAM

**JUNE 16-06-2015**

**Introduction GIT**

**A version control system (VCS**) is a methodology or tool that helps you keep track of changes you make to the files in your project. In its simplest, manual form, a VCS is you creating a copy of the file you’re working with and adding the date and time to the end of it.

**The repository** is the place where the version control system keeps track of all the changes you make. Most VCSs store the current state of the code, along with when each change was made, who made it, and a text log message that explains why they made the change.

**A fork is** a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

## GIT COMMANDS

We divide Git into high level ("porcelain") commands and low level ("plumbing") commands.

## High-level commands (porcelain)

We separate the porcelain commands into the main commands and some ancillary user utilities.

### Main porcelain commands

[git-add(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-add.html):- Add file contents to the index.

[git-am(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-am.html):- Apply a series of patches from a mailbox.

[git-archive(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-archive.html) :-Create an archive of files from a named tree.

[git-bisect(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-bisect.html) :-Find by binary search the change that introduced a bug.

[git-branch(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-branch.html) :-List, create, or delete branches.

[git-bundle(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-bundle.html) :-Move objects and refs by archive.

[git-checkout(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-checkout.html) :-Checkout a branch or paths to the working tree.

[git-cherry-pick(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-cherry-pick.html) :-Apply the changes introduced by some existing commits.

[git-citool(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-citool.html) :-Graphical alternative to git-commit.

[git-clean(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-clean.html) :-Remove untracked files from the working tree.

[git-clone(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-clone.html) :-Clone a repository into a new directory.

[git-commit(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-commit.html) :-Record changes to the repository.

[git-describe(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-describe.html) :-Show the most recent tag that is reachable from a commit.

[git-diff(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-diff.html) :-Show changes between commits, commit and working tree, etc.

[git-fetch(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-fetch.html) :-Download objects and refs from another repository.

[git-format-patch(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-format-patch.html) :-Prepare patches for e-mail submission.

[git-gc(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-gc.html) :-Cleanup unnecessary files and optimize the local repository.

[git-grep(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-grep.html) :-Print lines matching a pattern.

[git-gui(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-gui.html) :-A portable graphical interface to Git.

[git-init(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-init.html) :-Create an empty Git repository or reinitialize an existing one.

[git-log(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-log.html) :-Show commit logs.

[git-merge(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-merge.html) :-Join two or more development histories together.

[git-mv(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-mv.html) :-Move or rename a file, a directory, or a symlink.

[git-notes(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-notes.html) :-Add or inspect object notes.

[git-pull(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-pull.html) :-Fetch from and integrate with another repository or a local branch.

[git-push(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-push.html) :-Update remote refs along with associated objects.

[git-rebase(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-rebase.html) :-Forward-port local commits to the updated upstream head.

[git-reset(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-reset.html) :-Reset current HEAD to the specified state.

[git-revert(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-revert.html) :-Revert some existing commits.

[git-rm(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-rm.html) :-Remove files from the working tree and from the index.

[git-shortlog(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-shortlog.html) :-Summarize git log output.

[git-show(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-show.html) :-Show various types of objects.

[git-stash(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-stash.html) :-Stash the changes in a dirty working directory away.

[git-status(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-status.html) :-Show the working tree status.

[git-submodule(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-submodule.html) :-Initialize, update or inspect submodules.

[git-tag(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-tag.html) :-Create, list, delete or verify a tag object signed with GPG.

[gitk(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\gitk.html) :-The Git repository browser.

### Ancillary Commands

Manipulators:

[git-config(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-config.html) :-Get and set repository or global options.

[git-fast-export(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-fast-export.html) :-Git data exporter.

[git-fast-import(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-fast-import.html) L:-Backend for fast Git data importers.

[git-filter-branch(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-filter-branch.html) :-Rewrite branches.

[git-mergetool(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-mergetool.html) :-Run merge conflict resolution tools to resolve merge conflicts.

[git-pack-refs(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-pack-refs.html) :-Pack heads and tags for efficient repository access.

[git-prune(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-prune.html) :-Prune all unreachable objects from the object database.

[git-reflog(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-reflog.html) :-Manage reflog information.

[git-relink(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-relink.html) :-Hardlink common objects in local repositories.

[git-remote(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-remote.html) :-Manage set of tracked repositories.

[git-repack(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-repack.html) :-Pack unpacked objects in a repository.

[git-replace(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-replace.html) :-Create, list, delete refs to replace objects.

Interrogators:

[git-annotate(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-annotate.html) :-Annotate file lines with commit information.

[git-blame(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-blame.html) :-Show what revision and author last modified each line of a file.

[git-cherry(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-cherry.html) :-Find commits yet to be applied to upstream.

[git-count-objects(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-count-objects.html) :-Count unpacked number of objects and their disk consumption.

[git-difftool(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-difftool.html) :-Show changes using common diff tools.

[git-fsck(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-fsck.html) :-Verifies the connectivity and validity of the objects in the database.

[git-get-tar-commit-id(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-get-tar-commit-id.html) :-Extract commit ID from an archive created using git-archive.

[git-help(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-help.html) :-Display help information about Git.

[git-instaweb(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-instaweb.html) :-Instantly browse your working repository in gitweb.

[git-merge-tree(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-merge-tree.html) :-Show three-way merge without touching index.

[git-rerere(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-rerere.html) :-Reuse recorded resolution of conflicted merges.

[git-rev-parse(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-rev-parse.html) :-Pick out and massage parameters.

[git-show-branch(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-show-branch.html) :-Show branches and their commits.

[git-verify-tag(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-verify-tag.html):-Check the GPG signature of tags.

[git-whatchanged(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-whatchanged.html) :-Show logs with difference each commit introduces.

[gitweb(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\gitweb.html) :-Git web interface (web frontend to Git repositories).

### Interacting with Others

These commands are to interact with foreign SCM and with other people via patch over e-mail.

[git-archimport(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-archimport.html) :-Import an Arch repository into Git.

[git-cvsexportcommit(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-cvsexportcommit.html) :-Export a single commit to a CVS checkout.

[git-cvsimport(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-cvsimport.html) :-Salvage your data out of another SCM people love to hate.

[git-cvsserver(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-cvsserver.html):-A CVS server emulator for Git.

[git-imap-send(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-imap-send.html) :-Send a collection of patches from stdin to an IMAP folder.

[git-p4(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-p4.html) :-Import from and submit to Perforce repositories.

[git-quiltimport(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-quiltimport.html):-Applies a quilt patchset onto the current branch.

[git-request-pull(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-request-pull.html) :-Generates a summary of pending changes.

[git-send-email(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-send-email.html) :-Send a collection of patches as emails.

[git-svn(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-svn.html) :-Bidirectional operation between a Subversion repository and Git.

## Low-level commands (plumbing)

Although Git includes its own porcelain layer, its low-level commands are sufficient to support development of alternative porcelains. Developers of such porcelains might start by reading about [git-update-index(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-update-index.html) and [git-read-tree(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-read-tree.html).

The interface (input, output, set of options and the semantics) to these low-level commands are meant to be a lot more stable than Porcelain level commands, because these commands are primarily for scripted use. The interface to Porcelain commands on the other hand are subject to change in order to improve the end user experience.

The following description divides the low-level commands into commands that manipulate objects (in the repository, index, and working tree), commands that interrogate and compare objects, and commands that move objects and references between repositories.

### Manipulation commands

[git-apply(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-apply.html) :-Apply a patch to files and/or to the index.

[git-checkout-index(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-checkout-index.html) :-Copy files from the index to the working tree.

[git-commit-tree(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-commit-tree.html) ;-Create a new commit object.

[git-hash-object(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-hash-object.html) :Compute object ID and optionally creates a blob from a file.

[git-index-pack(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-index-pack.html) :-Build pack index file for an existing packed archive.

[git-merge-file(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-merge-file.html) :-Run a three-way file merge.

[git-merge-index(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-merge-index.html):-Run a merge for files needing merging.

[git-mktag(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-mktag.html) :-Creates a tag object.

[git-mktree(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-mktree.html) :-Build a tree-object from ls-tree formatted text.

[git-pack-objects(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-pack-objects.html) L:-Create a packed archive of objects.

[git-prune-packed(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-prune-packed.html) :-Remove extra objects that are already in pack files.

[git-read-tree(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-read-tree.html) :-Reads tree information into the index.

[git-symbolic-ref(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-symbolic-ref.html) :-Read, modify and delete symbolic refs.

[git-unpack-objects(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-unpack-objects.html) :-Unpack objects from a packed archive.

[git-update-index(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-update-index.html) :-Register file contents in the working tree to the index.

[git-update-ref(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-update-ref.html) :-Update the object name stored in a ref safely.

[git-write-tree(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-write-tree.html) :-Create a tree object from the current index.

### Interrogation commands

[git-cat-file(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-cat-file.html) :-Provide content or type and size information for repository objects.

[git-diff-files(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-diff-files.html) :-Compares files in the working tree and the index.

[git-diff-index(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-diff-index.html) :-Compare a tree to the working tree or index.

[git-diff-tree(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-diff-tree.html) :-Compares the content and mode of blobs found via two tree objects.

[git-for-each-ref(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-for-each-ref.html) :-Output information on each ref.

[git-ls-files(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-ls-files.html) :-Show information about files in the index and the working tree.

[git-ls-remote(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-ls-remote.html) :-List references in a remote repository.

[git-ls-tree(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-ls-tree.html) :-List the contents of a tree object.

[git-merge-base(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-merge-base.html) :-Find as good common ancestors as possible for a merge.

[git-name-rev(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-name-rev.html) :-Find symbolic names for given revs.

[git-pack-redundant(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-pack-redundant.html) :-Find redundant pack files.

[git-rev-list(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-rev-list.html) :-Lists commit objects in reverse chronological order.

[git-show-index(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-show-index.html) :-Show packed archive index.

[git-show-ref(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-show-ref.html) :-List references in a local repository.

[git-unpack-file(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-unpack-file.html) :-Creates a temporary file with a blob’s contents.

[git-var(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-var.html) :-Show a Git logical variable.

[git-verify-pack(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-verify-pack.html) :-Validate packed Git archive files.

In general, the interrogate commands do not touch the files in the working tree.

### Synching repositories

[git-daemon(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-daemon.html) :-A really simple server for Git repositories.

[git-fetch-pack(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-fetch-pack.html) :-Receive missing objects from another repository.

[git-http-backend(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-http-backend.html) :-Server side implementation of Git over HTTP.

[git-send-pack(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-send-pack.html) :-Push objects over Git protocol to another repository.

[git-update-server-info(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-update-server-info.html) :-Update auxiliary info file to help dumb servers.

The following are helper commands used by the above; end users typically do not use them directly.

[git-http-fetch(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-http-fetch.html) :-Download from a remote Git repository via HTTP.

[git-http-push(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-http-push.html) :-Push objects over HTTP/DAV to another repository.

[git-parse-remote(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-parse-remote.html) :-Routines to help parsing remote repository access parameters.

[git-receive-pack(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-receive-pack.html)

[git-receive-pack(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-receive-pack.html) :- Receive what is pushed into the repository.

[git-shell(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-shell.html) :-Restricted login shell for Git-only SSH access.

[git-upload-archive(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-upload-archive.html) :-Send archive back to git-archive.

[git-upload-pack(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-upload-pack.html) :-Send objects packed back to git-fetch-pack.

### Internal helper commands

These are internal helper commands used by other commands; end users typically do not use them directly.

[git-check-attr(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-check-attr.html) :-Display gitattributes information.

[git-check-ignore(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-check-ignore.html) :-Debug gitignore / exclude files.

[git-check-mailmap(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-check-mailmap.html) :-Show canonical names and email addresses of contacts.

[git-check-ref-format(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-check-ref-format.html) :-Ensures that a reference name is well formed.

[git-column(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-column.html) :-Display data in columns.

[git-credential(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-credential.html) :-Retrieve and store user credentials.

[git-credential-cache(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-credential-cache.html) :-Helper to temporarily store passwords in memory.

[git-credential-store(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-credential-store.html) :-Helper to store credentials on disk.

[git-fmt-merge-msg(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-fmt-merge-msg.html) :-Produce a merge commit message.

[git-mailinfo(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-mailinfo.html) ;-Extracts patch and authorship from a single e-mail message.

[git-mailsplit(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-mailsplit.html) :-Simple UNIX mbox splitter program.

[git-merge-one-file(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-merge-one-file.html) :-The standard helper program to use with git-merge-index.

[git-patch-id(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-patch-id.html) :-Compute unique ID for a patch.

[git-sh-i18n(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-sh-i18n.html) :-Git’s i18n setup code for shell scripts.

[git-sh-setup(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-sh-setup.html) :-Common Git shell script setup code.

[git-stripspace(1)](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\git-stripspace.html) :-Remove unnecessary whitespace.

# Everyday GIT With 20 Commands and above

**Individual Developer (Standalone)**

A standalone individual developer does not exchange patches with other people, and works alone in a single repository, using the following commands.

* [*git-init()*](https://www.kernel.org/pub/software/scm/git/docs/git-init.html) to create a new repository.
* [*git-show-branch()*](https://www.kernel.org/pub/software/scm/git/docs/git-show-branch.html) to see where you are.
* [*git-log()*](https://www.kernel.org/pub/software/scm/git/docs/git-log.html)to see what happened.
* [*git-checkout()*](https://www.kernel.org/pub/software/scm/git/docs/git-checkout.html) and [git-branch()](https://www.kernel.org/pub/software/scm/git/docs/git-branch.html) to switch branches.
* [*git-add()*](https://www.kernel.org/pub/software/scm/git/docs/git-add.html) to manage the index file.
* [*git-diff()*](https://www.kernel.org/pub/software/scm/git/docs/git-diff.html) and [*git-status()*](https://www.kernel.org/pub/software/scm/git/docs/git-status.html) to see what you are in the middle of doing.
* [*git-commit()*](https://www.kernel.org/pub/software/scm/git/docs/git-commit.html)to advance the current branch.
* [*git-reset()*](https://www.kernel.org/pub/software/scm/git/docs/git-reset.html) and [*git-checkout()*](https://www.kernel.org/pub/software/scm/git/docs/git-checkout.html) (with pathname parameters) to undo changes.
* [*git-merge()*](https://www.kernel.org/pub/software/scm/git/docs/git-merge.html) to merge between local branches.
* [*git-rebase()*](https://www.kernel.org/pub/software/scm/git/docs/git-rebase.html)to maintain topic branches.
* [*git-tag()*](https://www.kernel.org/pub/software/scm/git/docs/git-tag.html)to mark known point.

## Individual Developer (Participant)

A developer working as a participant in a group project needs to learn how to communicate with others, and uses these commands in addition to the ones needed by a standalone developer.

* [*git-clone()*](https://www.kernel.org/pub/software/scm/git/docs/git-clone.html)from the upstream to prime your local repository.
* [*git-pull()*](https://www.kernel.org/pub/software/scm/git/docs/git-pull.html)and [git-fetch()](https://www.kernel.org/pub/software/scm/git/docs/git-fetch.html)from "origin" to keep up-to-date with the upstream.
* [git-push()](https://www.kernel.org/pub/software/scm/git/docs/git-push.html) to shared repository, if you adopt CVS style shared repository workflow.
* [git-format-patch()](https://www.kernel.org/pub/software/scm/git/docs/git-format-patch.html) to prepare e-mail submission, if you adopt Linux kernel-style public forum workflow.

## Integrator

A fairly central person acting as the integrator in a group project receives changes made by others, reviews and integrates them and publishes the result for others to use, using these commands in addition to the ones needed by participants.

* [***git-am()***](https://www.kernel.org/pub/software/scm/git/docs/git-am.html) to apply patches e-mailed in from your contributors.
* [*git-pull()*](https://www.kernel.org/pub/software/scm/git/docs/git-pull.html) to merge from your trusted lieutenants.
* [*git-format-patch(*)](https://www.kernel.org/pub/software/scm/git/docs/git-format-patch.html) to prepare and send suggested alternative to contributors.
* [*git-revert()*](https://www.kernel.org/pub/software/scm/git/docs/git-revert.html) *to* undo botched commits.
* [*git-push(*)](https://www.kernel.org/pub/software/scm/git/docs/git-push.html) to publish the bleeding edge.

## Repository Administration

A repository administrator uses the following tools to set up and maintain access to the repository by developers.

* [*git-daemon()*](https://www.kernel.org/pub/software/scm/git/docs/git-daemon.html) to allow anonymous download from repository.
* [*git-shell()*](https://www.kernel.org/pub/software/scm/git/docs/git-shell.html) can be used as a restricted login shell for shared central repository users.
* [Update hook howto](file:///C:\Program%20Files%20%28x86%29\Git\doc\git\html\howto\update-hook-example.html) has a good example of managing a shared central repository.

## Set Up Git

## Instillation Git

1. On your computer, open the **Git Shell** application.

## Write git config --global user.name”INSA”

## Write git config --global user.email [INSA@gmail.com](mailto:INSA@gmail.com)

GitHub: - is a web-based hosting service for software development projects that uses the Git revision control system

**Common command Basic use Git**

* Change some files
* See what you've changed

git status

git diff

git log

* Indicate what changes to save

*git add*

* Commit to those changes

*git commit*

* Push the changes to GitHub

*git push*

Pull changes from your collaborator

*git push*

### Authenticating with GitHub from Git

When you connect to a GitHub repository from Git, you'll need to authenticate with GitHub using either HTTPS or SSH.

#### Connecting over HTTPS (recommended)

If you [clone with HTTPS](https://help.github.com/articles/which-remote-url-should-i-use#cloning-with-https-recommended), you can [cache your GitHub password in Git](https://help.github.com/articles/caching-your-github-password-in-git) using a **credential helper**.

#### Connecting over SSH

If you [clone with SSH](https://help.github.com/articles/which-remote-url-should-i-use#cloning-with-ssh), you must [generate SSH keys](https://help.github.com/articles/generating-ssh-keys) on each computer you use to push or pull from GitHub.