Installed see ok ?

Git config –list

core.symlinks=false

core.autocrlf=true

core.editor=gitpad

color.diff=auto

color.status=auto

color.branch=auto

color.interactive=true

color.ui=true

pack.packsizelimit=2g

help.format=html

http.sslcainfo=/bin/curl-ca-bundle.crt

sendemail.smtpserver=/bin/msmtp.exe

diff.astextplain.textconv=astextplain

rebase.autosquash=true

credential.helper=!github --credentials

filter.ghcleansmudge.clean=cat

filter.ghcleansmudge.smudge=cat

push.default=upstream

alias.dt=difftool

alias.mt=mergetool

diff.tool=vs11

difftool.prompt=false

difftool.bc4.cmd="c:/program files (x86)/beyond compare 3/bcomp.exe" "$LOCAL" "$REMOTE"

difftool.p4.cmd="c:/program files/Perforce/p4merge.exe" "$LOCAL" "$REMOTE"

difftool.vs11.cmd="c:/program files (x86)/microsoft visual studio 11.0/common7/ide/devenv.exe" '//diff' "$LOCAL" "$REMOTE"

merge.tool=bc3

mergetool.prompt=false

mergetool.keepbackup=false

mergetool.bc3.cmd="c:/program files (x86)/beyond compare 3/bcomp.exe" "$LOCAL" "$REMOTE" "$BASE" "$MERGED"

mergetool.bc3.trustexitcode=true

mergetool.p4.cmd="c:/program files/Perforce/p4merge.exe" "$BASE" "$LOCAL" "$REMOTE" "$MERGED"

mergetool.p4.trustexitcode=false

remote.origin.fetch=+refs/heads/\*:refs/remotes/origin/\*

remote.origin.fetch=+refs/pull/\*/head:refs/remotes/origin/pr/\*

user.name=siriwardenap

user.email=siriwardenap@gmail.com

core.repositoryformatversion=0

core.filemode=false

core.bare=false

core.logallrefupdates=true

core.symlinks=false

core.ignorecase=true

core.hidedotfiles=dotGitOnly

gui.wmstate=zoomed

gui.geometry=887x427+337+63 171 192

Git has three main states that your files can reside in:

committed,

modified,

staged.

Committed means that the data is safely stored in your local database.

Modified means that you have changed the file but have not committed it to your database yet.

Staged means that you have marked a modified file in its current version to go into your next commit snapshot.

This leads us to the three main sections of a Git project:

the Git directory,

the working directory

the staging area.



The Git directory is where Git stores the metadata and object database for your

project. This is the most important part of Git, and it is what is copied when you clone

a repository from another computer.

The working directory is a single checkout of one version of the project. These files

are pulled out of the compressed database in the Git directory and placed on disk for

you to use or modify.

The staging area is a simple file, generally contained in your Git directory, that

stores information about what will go into your next commit. It’s sometimes referred

to as the index, but it’s becoming standard to refer to it as the staging area.

The basic Git workflow goes something like this:

1. You modify files in your working directory.

2. You stage the files, adding snapshots of them to your staging area.

3. You do a commit, which takes the files as they are in the staging area and stores

that snapshot permanently to your Git directory.

If a particular version of a file is in the git directory, it’s considered committed. If it’s

modified but has been added to the staging area, it is staged. And if it was changed

since it was checked out but has not been staged, it is modified. In Chapter 2, you’ll

learn more about these states and how you can either take advantage of them or skip

the staged part entirely.

Git - -help

usage: git [--version] [--help] [-c name=value]

[--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]

[-p|--paginate|--no-pager] [--no-replace-objects] [--bare]

[--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]

<command> [<args>]

The most commonly used git commands are:

add Add file contents to the index

bisect Find by binary search the change that introduced a bug

branch List, create, or delete branches

checkout Checkout a branch or paths to the working tree

clone Clone a repository into a new directory

commit Record changes to the repository

diff Show changes between commits, commit and working tree, etc

fetch Download objects and refs from another repository

grep Print lines matching a pattern

init Create an empty Git repository or reinitialize an existing one

log Show commit logs

merge Join two or more development histories together

mv Move or rename a file, a directory, or a symlink

pull Fetch from and merge with another repository or a local branch

push Update remote refs along with associated objects

rebase Forward-port local commits to the updated upstream head

reset Reset current HEAD to the specified state

rm Remove files from the working tree and from the index

show Show various types of objects

status Show the working tree status

tag Create, list, delete or verify a tag object signed with GPG

'git help -a' and 'git help -g' lists available subcommands and some

concept guides. See 'git help <command>' or 'git help <concept>'

to read about a specific subcommand or concept.

Git uses default edito VI / VIM Emacs

Git uses default diff tool to resolve merge conflicts

If different say vimdiff needed then

> git config - - global merge.tool vimdiff does it

* Git config user.name ----🡪 siriwardenap

Git help config ---🡪 full help page on config //ly on any command

configure and initialize a repository,

begin and stop tracking files, and stage and commitchanges.

We’ll also show you how to set up Git to ignore certain files and file patterns,

how to undo mistakes quickly and easily,

how to browse the history of your project and view changes between commits,

how to push and pull from remote repositories

**Getting Repository**

A - takes an existing project or directory and imports it into Git.

B - clones an existing Git repository from another server.

**How to start tracking an existing project**

go to the project’s directory and type

$ git init

**to start version-controlling existing files** (as opposed to an empty directory),

you should probably begin tracking those files and do an initial commit. You can

accomplish that with a few git add commands that specify the files you want to track,

followed by a commit:

$ git add \*.c

$ git add README

$ git commit m ’initial project version’