



Persistent

**Git:**

# Working with Git Local

Persistent University



## Key Learning Points

- o **Creating local repository, adding files, and committing changes**
- o **Viewing log and diffs**
- o **Staging changes as multiple changes**
- o **Deleting and renaming files**
- o **Ignoring Files**
- o **Undoing/redoin changes to the local copy and repository**
- o **Cleaning the working copy**

# Git Local Operations

A decorative orange line graphic that starts as a horizontal line from the left edge, crosses the title, and then turns 90 degrees clockwise into a vertical line. A circle is drawn in the upper right quadrant, with its center at the intersection of the horizontal and vertical lines.

## Configuring Git: Simple Command Line

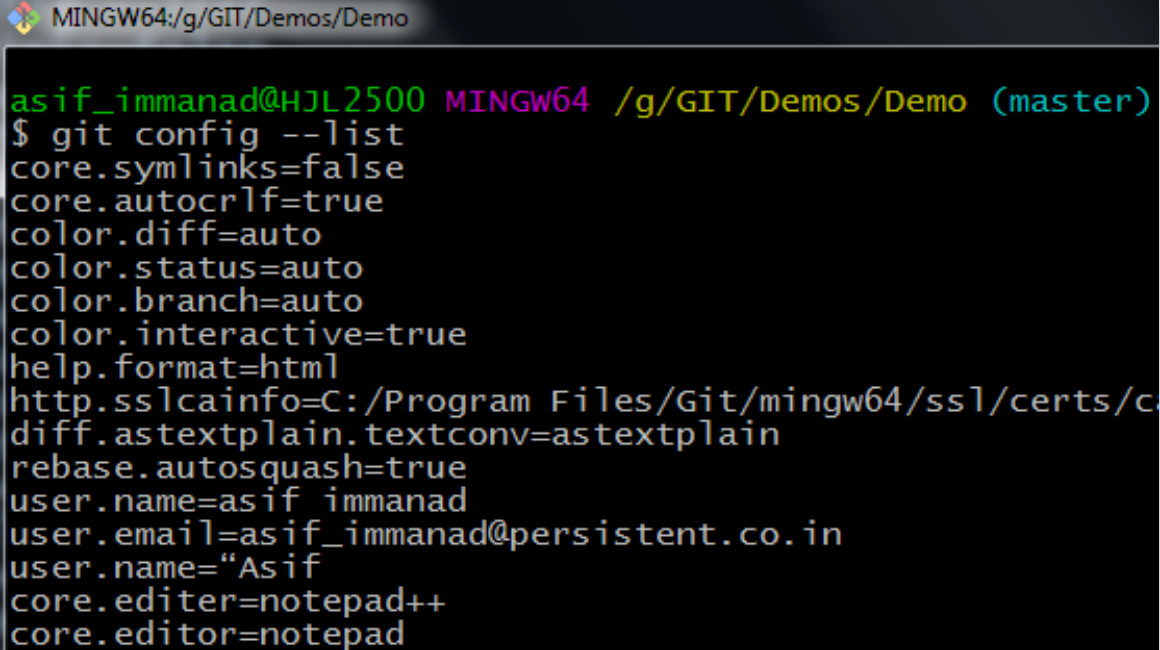
- Open Git Bash Terminal
- Where am I??
  - pwd – print working directory
- How do I move around??
  - cd – change directory
  - cd c:/csm\_classes – change to a specific path
  - cd .. – change to parent directory
- What files are there?
  - ls – list
  - ls -l – list with details

```
$ pwd
```

```
/c/Users/asif_immanad
```

## Tell Git who you are

- Update your configuration, one time only
  - git config – Global user.name “Asif Immanad”
  - git config – Global user.email [asif\\_immanad@persistent.co.in](mailto:asif_immanad@persistent.co.in)
  - git config – Global core.editor notepad++
  - git config – List (or git config –l)

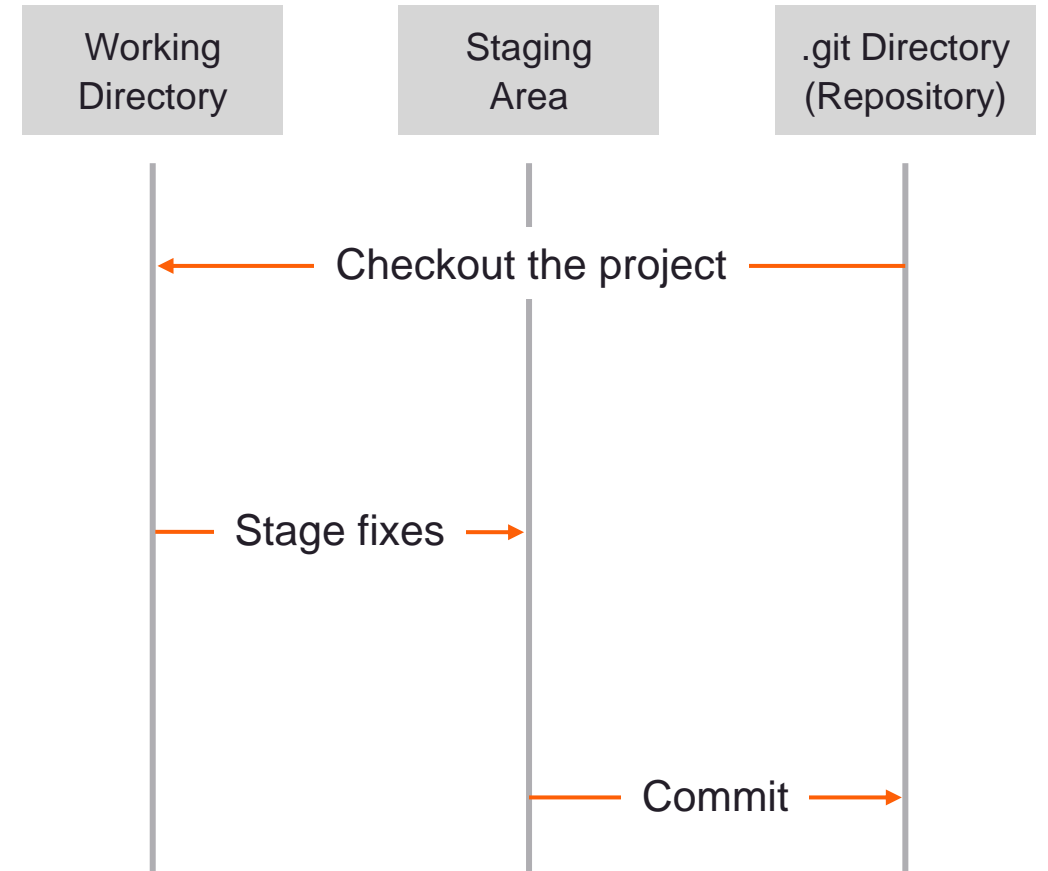
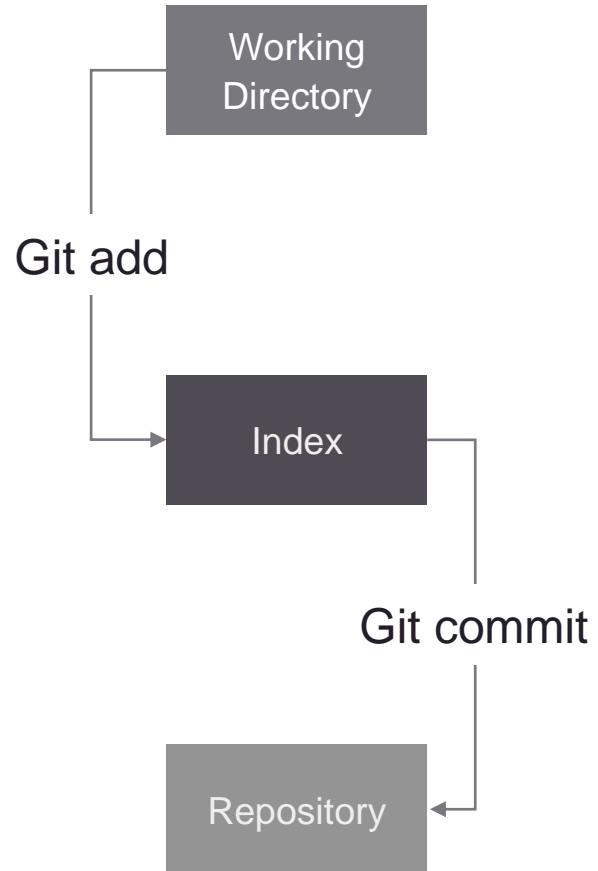
A terminal window titled 'MINGW64:/g/GIT/Demos/Demo' showing the output of the 'git config --list' command. The output lists various Git configuration settings, including user.name, user.email, core.editor, and color options. The user.name is 'asif\_immanad' and the user.email is 'asif\_immanad@persistent.co.in'. The core.editor is set to 'notepad++'. The terminal text is as follows:

```
asif_immanad@HJL2500 MINGW64 /g/GIT/Demos/Demo (master)
$ git config --list
core.symlinks=false
core.autocrlf=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
help.format=html
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/c
diff.astextplain.textconv=astextplain
rebase.autosquash=true
user.name=asif_immanad
user.email=asif_immanad@persistent.co.in
user.name="Asif"
core.editor=notepad++
core.editor=notepad
```

## Workflow of Git

### A Basic Workflow

- Edit files
- Stage the changes
- Review your changes
- Commit the changes



## Basic Steps

- Configure Git
- Initialize Git
- Add files – Adds to Staging area
- Commit – Log everything to repository
- Check Status – View what you are up to
- View diff – View the difference between two states
- View log – View all commit log

## Getting Started

- Create a Java Project (mine is GitIntro).
- cd into project directory.
- Use git init command to initialize the repository.
- Create a class (mine is HelloGit).
- This is one-time process.

```
crader_a@EECS-LAPTOP-02 /c/csm_classes/cs306/JavaCode/Practice
$ cd GitIntro

crader_a@EECS-LAPTOP-02 /c/csm_classes/cs306/JavaCode/Practice/GitIntro
$ ls
bin  src

crader_a@EECS-LAPTOP-02 /c/csm_classes/cs306/JavaCode/Practice/GitIntro
$ git init
Initialized empty Git repository in c:/csm_classes/cs306/JavaCode/Practice/GitIntro/.git/
```



## Ignoring files

- It's important to tell Git what files you do *not* want to track
- Temp files, executable files, etc. do not need version control (and can cause major issues when merging!)
- <https://help.github.com/articles/ignoring-files>
- Example (place in root of repo):
  - \*.class
  - .project
  - .classpath
  - .settings/

## More .gitignore files

### Example of .gitignore files :-

- **# Directories #**

- /build/
- /bin/

- **# OS Files #**

- .DS\_Store
- \*.class

- **# Package Files #**

- \*.jar
- \*.war
- \*.ear

- **# Windows image file caches**

- Thumbs.db

- **# Folder config file**

- Desktop.ini

- **# OSX**

- .DS\_Store
- .svn

- **# Eclipse**

- .project
- .metadata
- bin/\*\*
- tmp/\*\*
- \*.tmp
- local.properties
- .classpath
- .settings/
- .loadpath

## What's the Status?

- Once we have initiated the repository and made some changes on to the file, we may check the status.
- To check the status use Git status.

### Nothing tracked yet.

```
$ git status
On branch master

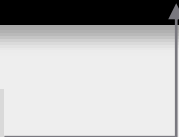
Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .gitignore
    src/

nothing added to commit but untracked files present (use "git add" to track)
```

Get used to reading helpful messages



## Let's Track a File

- Once we have done with making the changes in file, we may add them to repository to start tracking the file.
- To start tracking a file use Git add.

### Git add – Tells Git to track a file

```
$ git add src/*  
  
crader_a@EECS-LAPTOP-02 /c/csm_classes/cs306/JavaCode/Practice/GitIntro (master)  
  
$ git status  
On branch master  
  
Initial commit  
  
Changes to be committed:  
  (use "git rm --cached <file>..." to unstage)  
  
    new file:   src/HelloGit.java  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
  
    .gitignore
```

## Let's Track a File

- The Git add command adds a change in the working directory to the staging area.
- However, Git add doesn't really affect the repository in any significant way—changes are not actually recorded until you run Git commit.
- In conjunction with these commands, you'll also need Git status to view the state of the working directory and the staging area.
- Git add <file>
  - Stage all changes in <file> for the next commit.
- Git add <directory>
  - Stage all changes in <directory> for the next commit.

## Saving Changes, Let's Commit...

- When we feel the changes are ready to commit.
- The Git commit command commits the staged snapshot to the project history.
- Committed snapshots can be thought of as “safe” versions of a project — Git will never change them unless you explicitly ask it to.
- Along with Git add, this is one of the most important Git commands.
- To commit the changes we need to provide some message along.
  - `Git commit -m “<message>”`
- To add and commit as well we can use.
  - `Git commit -am “<message>”`

Commit the file...

When you commit, you must provide a comment  
(if you forget, Git will open a text editor so you can write one).

```
$ git commit -m "Initial project version"
[master eaf901e] Initial project version
1 file changed, 4 insertions(+)
create mode 100644 .gitignore

crader_a@EECS-LAPTOP-02 /c/csm_classes/cs306/JavaCode/Practice/GitIntro (master)

$ git status
On branch master
nothing to commit, working directory clean
```

## Let's Inspect...

- The Git status command displays the state of the working directory and the staging area.
- It lets you see which changes have been staged, which haven't, and which files aren't being tracked by Git.
- To inspect use:
  - Git status
- List which files are staged, unstaged, and untracked.



## What if you change the file?

```
$ 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:   src/HelloGit.java

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

Notice more helpful hints Git provides. You could add to the staging area OR add & commit in one step.

```
$ git commit -am "Saying hello to GIT"
[master 8a996be] Saying hello to GIT
1 file changed, 1 insertion(+), 1 deletion(-)
```

Be careful if you add to the staging area and then make more changes – The file can appear as both staged and unstaged. For now, we can use –am.

## You made some changes!!! But What?

- We have made some changes but not sure what we have made!!!
- Let's check the changes and compare with what we have in repository.
- To differentiate working version and committed version we can use:
  - Git diff
  - Show changes between commits, commit and working tree, etc.

You made some changes!!! But What?

This command compares your working directory with your staging area.  
These are the changes that are not yet staged.

```
$ git diff
diff --git a/src/HelloGit.java b/src/HelloGit.java
index edeff09..78ad014 100644
--- a/src/HelloGit.java
+++ b/src/HelloGit.java
@@ -3,6 +3,7 @@ public class HelloGit {

    public static void main(String[] args) {
        System.out.println("Hello Git!");
+       System.out.println("It's good to know you!");

    }
```

What if you've committed all your changes?

If **all the changes are committed**, using diff command doesn't show anything.

```
$ git commit -am "Nice message to git"
[master db07dba] Nice message to git
1 file changed, 1 insertion(+)

crader_a@EECS-LAPTOP-02 /c/csm_classes/cs306/JavaCode/Practice/GitIntro (master)
$ git diff
```

**diff** doesn't have anything to display.

## What if you remove a file?

If we remove a file from folder Git replies saying that file has deleted.

## File added not committed

```
$ git add src/HelloWorld.java
crader_a@EECS-LAPTOP-02 /c/csm_classes/cs306/JavaCode/Practice/GitIntro (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   src/HelloWorld.java
```

## Now I remove HelloWorld.java

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   src/HelloWorld.java

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    deleted:    src/HelloWorld.java
```

## What if you remove a file?

- To remove file from being tracked use:
  - `git rm <filename>`

```
$ git rm src/HelloWorld.java
rm 'src/HelloWorld.java'

crader_a@EECS-LAPTOP-02 /c/csm_classes/cs306/JavaCode/Practice/GitIntro (master)

$ git status
On branch master
nothing to commit, working directory clean
```

**This removes the file from being tracked. If you've already committed, the file is still in the database.**

## Worked whole day...but what all have I done?

- The Git log command displays committed snapshots.
- It lets you list the project history, filter it, and search for specific changes.
- While Git status lets you inspect the working directory and the staging area, Git log only operates on the committed history.
- So to go back and see what all we have done use:
  - Git log
- Display's the entire commit history using the default formatting.

So what all have I done?

```
$ git log
commit db07dba2ea40c7fc3f6c3ba5bac957621052341
Author: Cyndi Rader <crader@mines.edu>
Date:   Mon Jun 9 17:24:43 2014 -0600

    Nice message to git

commit 8a996be50f3d433b267d944c5062c1754034b4a7
Author: Cyndi Rader <crader@mines.edu>
Date:   Mon Jun 9 17:21:05 2014 -0600

    Saying hello to GIT

commit eaf901e7da8f5fb7afd0230aaa2c3dda44dd7d98
Author: Cyndi Rader <crader@mines.edu>
Date:   Mon Jun 9 17:19:13 2014 -0600

    Initial project version

commit a615db996eb8ead9078a8c0690610efa821248e2
Author: Cyndi Rader <crader@mines.edu>
Date:   Mon Jun 9 17:15:35 2014 -0600

    Initial version
```

There are many useful options for Git log.



## Difficulty in Finding Log?

- **git log -n <limit>**
  - git log -n 3 will display only 3 commits.
- **git log --oneline**
  - Condense each commit to a single line.
- **git log --author="<pattern>"**
  - Search for commits by a particular author. The <pattern> argument can be a plain string or a regular expression.
- **git log --grep="<pattern>"**
  - Search for commits with a commit message that matches <pattern>, which can be a plain string or a regular expression.
- **git log <since>..<until>**
  - Show only commits that occur between <since> and <until>.
- **git log <file>**
  - Only display commits that include the specified file. This is an easy way to see the history of a particular file.

## Git Log Options

### Can specify a format, such as:

- `git log --oneline`
- `git log --oneline -3`  
`<branchname>`
- `git log --pretty=oneline`

### Can filter, such as:

- `git log --since=2.weeks`
- includes filters like `--since`, `--after`, `--until`, `--before`, `--author`, etc.
- `git log --grep= "file"` → checks for matching commit message

### Can redirect output to file, such as:

- `git log >> gitlog.txt`

## Undoing Changes

- The git checkout command serves three distinct functions:
  - Checking out files,
  - Checking out commits, and
  - Checking out branches.
- Checking out a commit makes the entire working directory match that commit.
- View an old state without altering your current state in any way.
- Let's you see an old version of that particular file, leaving the rest of your working directory untouched.
- Use: `git checkout master`.
- Use: `git checkout a1e8fb5` – (a1e8fb5 is SHA key for commit).
- This makes your working directory match the exact state of the a1e8fb5 commit.

## Reverting

- The Git revert command undoes a committed snapshot.
- When you want to remove an entire commit from your project history.
- This can be useful, for example, if you're tracking down a bug and find that it was introduced by a single commit.
- Instead of manually going in, fixing it, and committing a new snapshot, you can use Git revert to automatically do all of this for you.
- Use: `git revert <commit>`

## Resetting

- If Git revert is a “safe” way to undo changes, Git reset is the dangerous method.
- There is no way to retrieve the original copy.
- It is a permanent undo.
- It’s one of the only Git commands that has the potential to lose your work.
- Git reset:
  - Reset the staging area to match the most recent commit but leave the working directory unchanged.
- Git reset <file>:
  - Remove the specified file from the staging area but leave the working directory unchanged.

## Reverting vs. Resetting

- Git revert undoes a single commit—it does not “revert” back to the previous state of a project by removing all subsequent commits.
- Reverting doesn’t change the project history.
- Git revert is able to target an individual commit at an arbitrary point in the history, whereas Git reset can only work backwards from the current commit.

## Un-stage and Rename File

- We can simply use a 'git mv' to rename a file where mv stands for move in Linux for renaming.  
eg.: `git mv filetodelete.txt keepfile.txt`.
- The way we stage a file to track, the same way we can unstage the file to untrack.  
eg.: `git reset HEAD filename.txt`.

## FAQs

- What is Version Controlling?
- What is SVN and its Pro's and Con's?
- History of Git?
- What is Git?
- Benefits of Git over other Version Controlling Tools?
- How Git works – Three States?



## Summary

- With this we have come to an end of our first session, where we discussed about.
  - What is Git?
  - Benefits and working of Git.
- At the end of this session, we see that you are now able to answer following questions:
  - What is Version Controlling?
  - How Git is Distributed Version Control?
  - What are the states in Git?
- In the next session we will discuss about.
  - Working with Git and Local Operations on Git.

## Reference Material: Websites & Blogs

- <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>
- <https://git-scm.com/video/what-is-git>
- [https://en.wikipedia.org/wiki/Git\\_\(software\)](https://en.wikipedia.org/wiki/Git_(software))
- <http://www.vogella.com/tutorials/Git/article.html>
- <https://www.siteground.com/tutorials/git/>

## Pro Git

- By Scott Chacon and Ben Straub
- Publisher: Apress

## Version Control with Git

- By Jon Loeliger, Matthew McCullough
- Publisher: O'Reilly Media

## Key Contacts

### Git Interactive

Vaishali Khatal

[vaishali\\_khatal@persistent.com](mailto:vaishali_khatal@persistent.com)

Asif Immanad

[asif\\_immanad@persistent.co.in](mailto:asif_immanad@persistent.co.in)

### Vice President

Shubhangi Kelkar

[shubhangi\\_kelkar@persistent.co.in](mailto:shubhangi_kelkar@persistent.co.in)



# Thank you!

Persistent University

