GIT

Session Objective

- Version Control Systems
- Types of Version Control System
- About Git
- Local Git areas
- Git workflow Life cycle
- Creating Git Repo
- Git commands
- Git commit
- Undo changes on file
- Viewing history and diffs
- Git Reset
- Git Revert
- Git Fetch
- Git Fetch vs Git Pull
- Branching & Merging
- Rebase
- Interaction with Remote Repo
- Stashing commits
- Git hosting services

Gamification

To make the participants familiarize with Git through activity.

Activity on Git

Version Control System

Version Control System

Version Control is a system that records changes to a file or set of files

The following are the types of version control systems:

- Local Version Control System - e.g. RCS
- Centralized Version Control System – e.g. CVS, Subversion, Perforce
- Distributed Version Control System – e.g. Git, Mercurial, Bazaar or Darcs

Why Version Control?

For working by yourself:

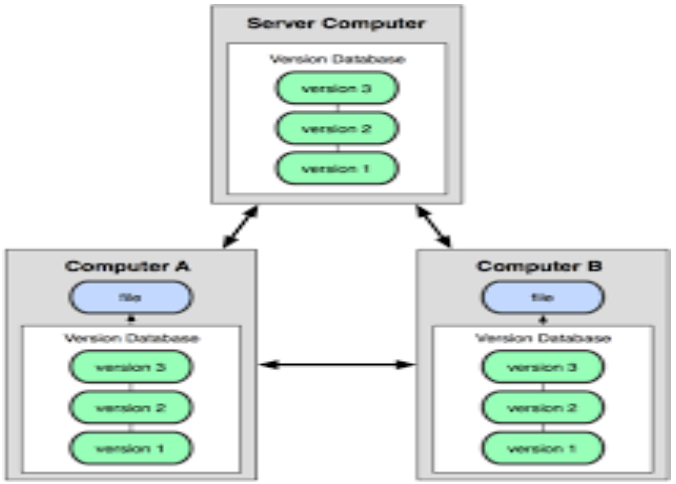
Gives you a “time machine” for going back to earlier versions
Gives you great support for different versions (standalone, web app, etc.) of the same basic project

For working with others:

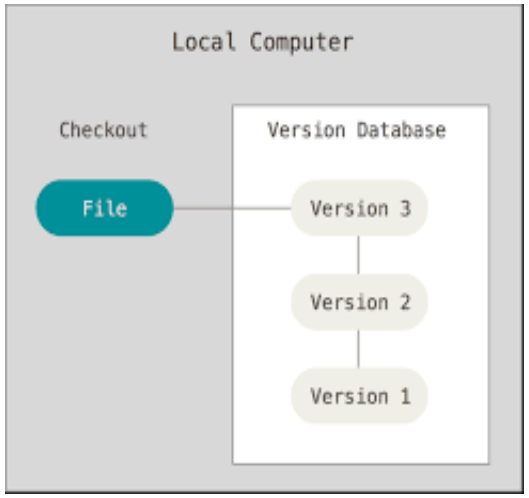
Greatly simplifies concurrent work, merging changes

Types of Version Control System

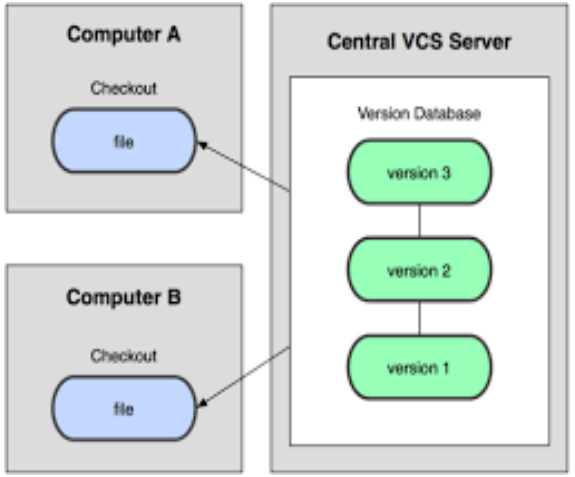
Distributed VCS



Local VCS



Centralized VCS

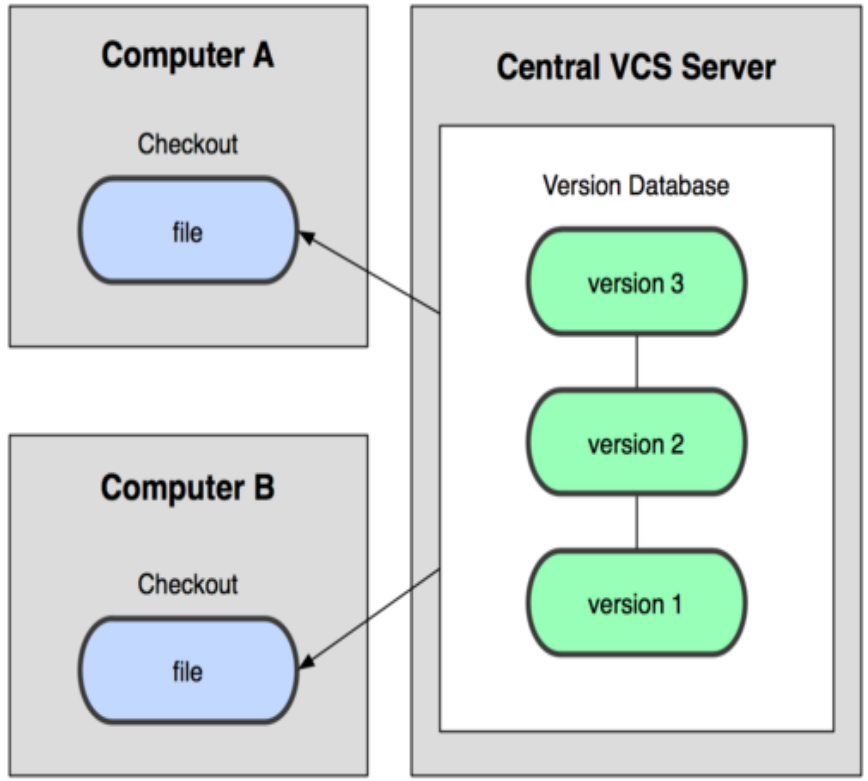


Centralized VCS

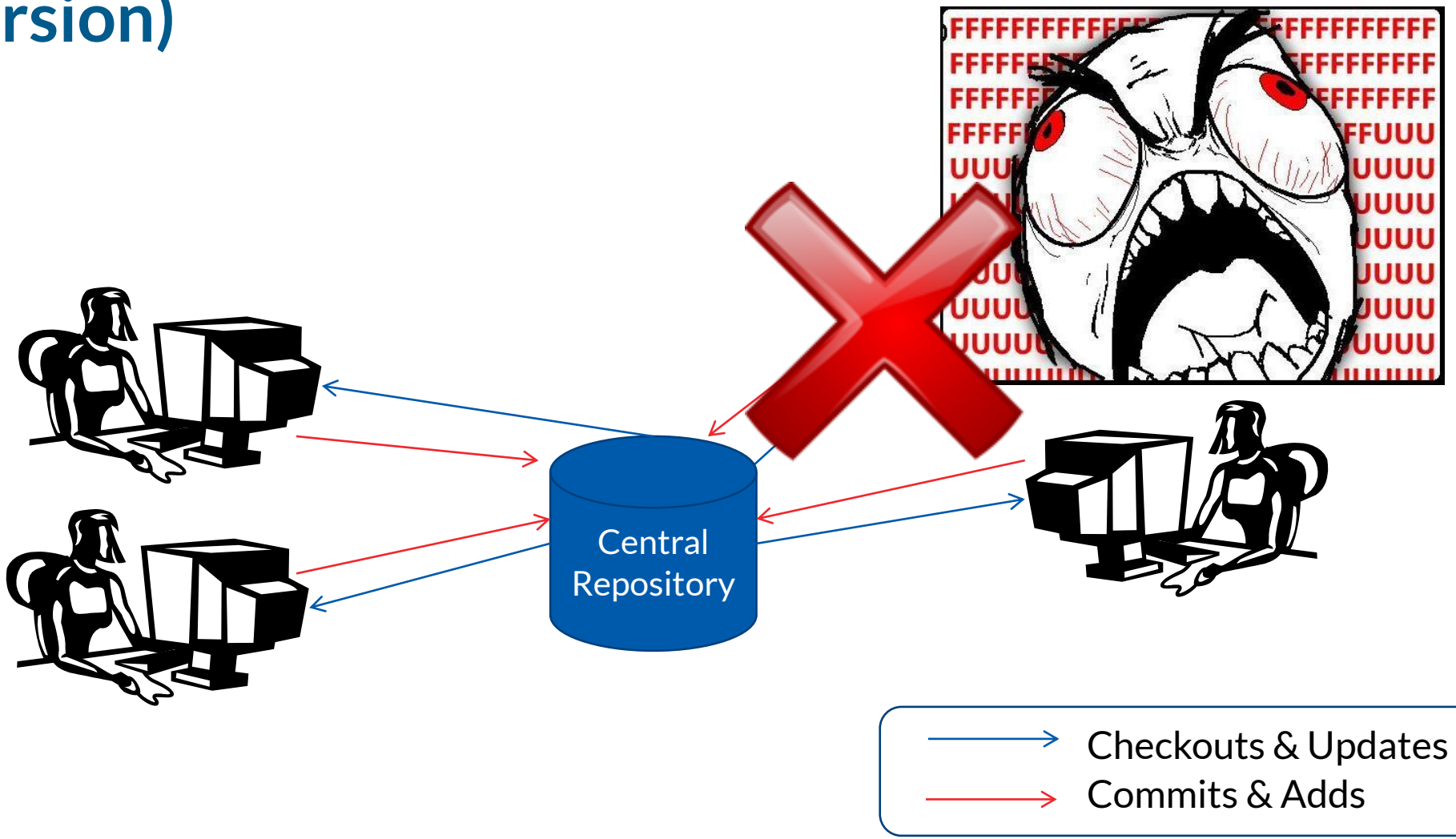
- In Subversion, CVS, Perforce, etc. A central server repository (repo) holds the "official copy" of the code – the server maintains the sole version history of the repo

- You make "checkouts" of it to your local copy
  - you make local modifications
  - your changes are not versioned

- When you're done, you "check in" back to the server
  - your check-in increments the repo's version



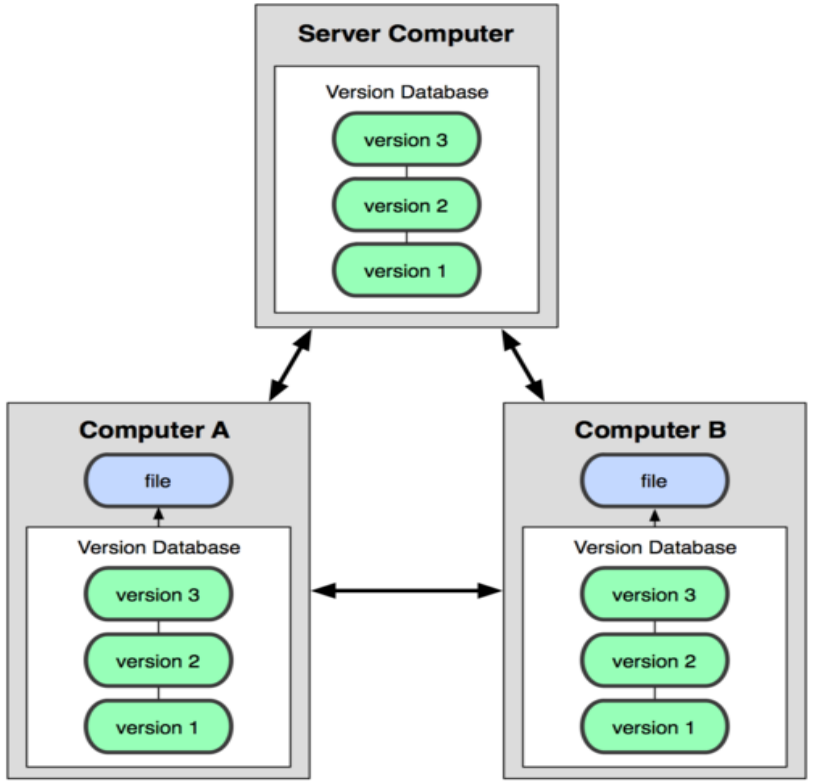
Traditional Centralized VC model (CVS, Subversion)



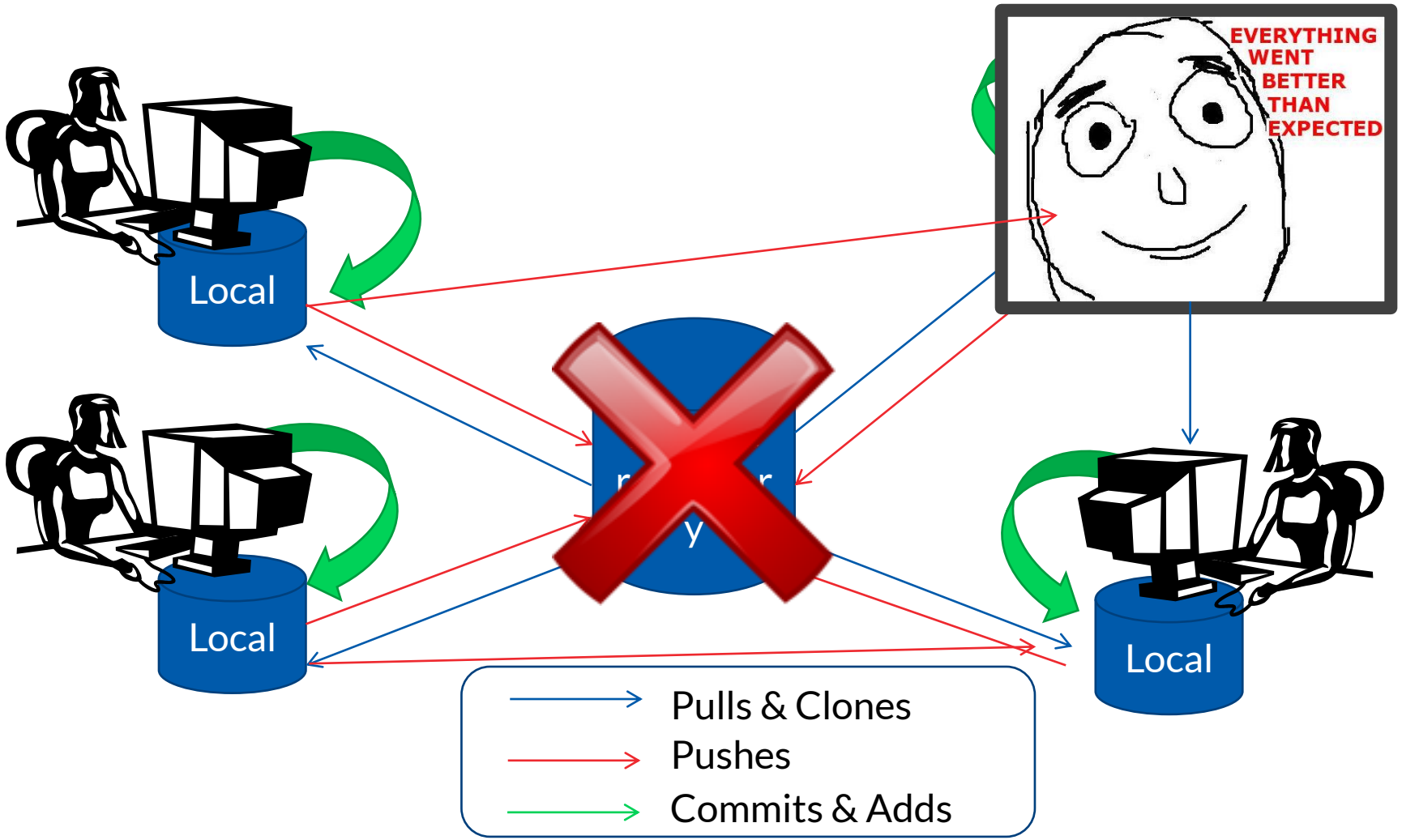
Distributed VCS

- Distributed version control system(DVCS) keeps track of software revisions and allows many developers to work on a given project without maintaining a connection to a common network.

- E.g., Git, Mercurial, Bazaar



Distributed version control system (git)



GIT

Git Basics

- Git is a free and open-source distributed version control system designed to handle everything from small to large projects with speed and efficiency.
- It was initially designed and developed by Linus Torvalds for Linux kernel development.
- Now it is maintained by Junio Hamano.
- Every Git working directory contains a full-fledged repository with complete history and full revision tracking capabilities.
- It is not dependent on network access or a central server.

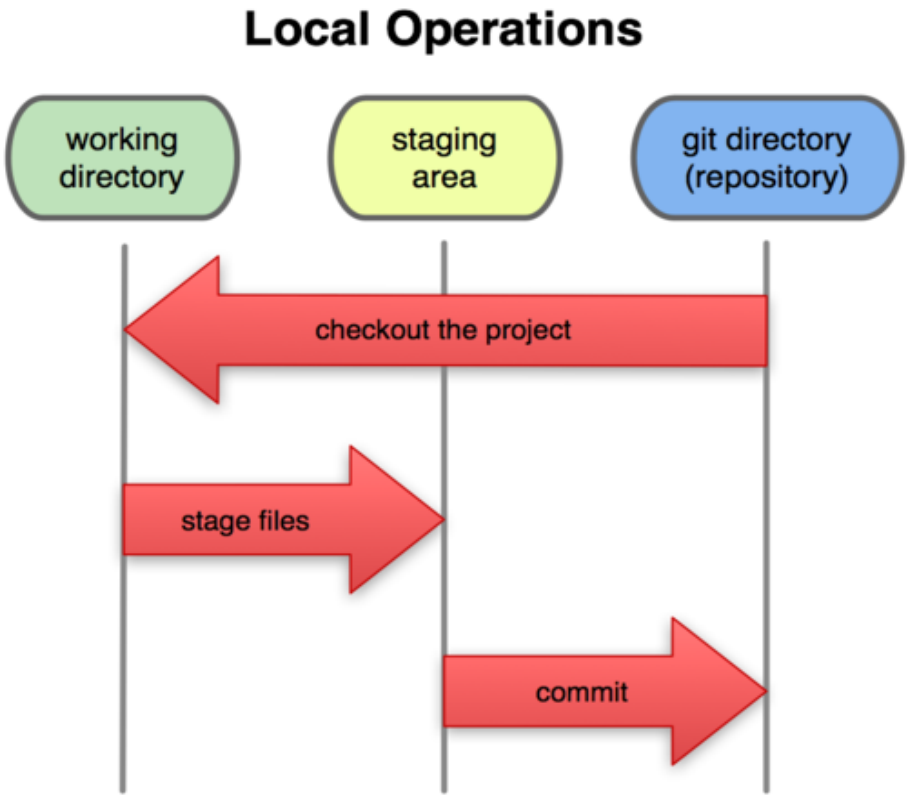
Goals of Git:

- Speed
- Support for non-linear development
  (thousands of parallel branches)
- Fully distributed
  (Committing code does not require access to a central server)
- Able to handle large projects efficiently
  (speed and data size) (13,5 million lines of code)

Local Git areas

Three main states of Git:

- Working Directory
- Staging Area
- Git Directory



Local Git Areas (Cont..)

Git Directory – Stores the metadata and object database for the project (while cloning the repository)

Working Directory – Single checkout of one version of the project.

The files in the directory are pulled out of the compressed database in the Git directory

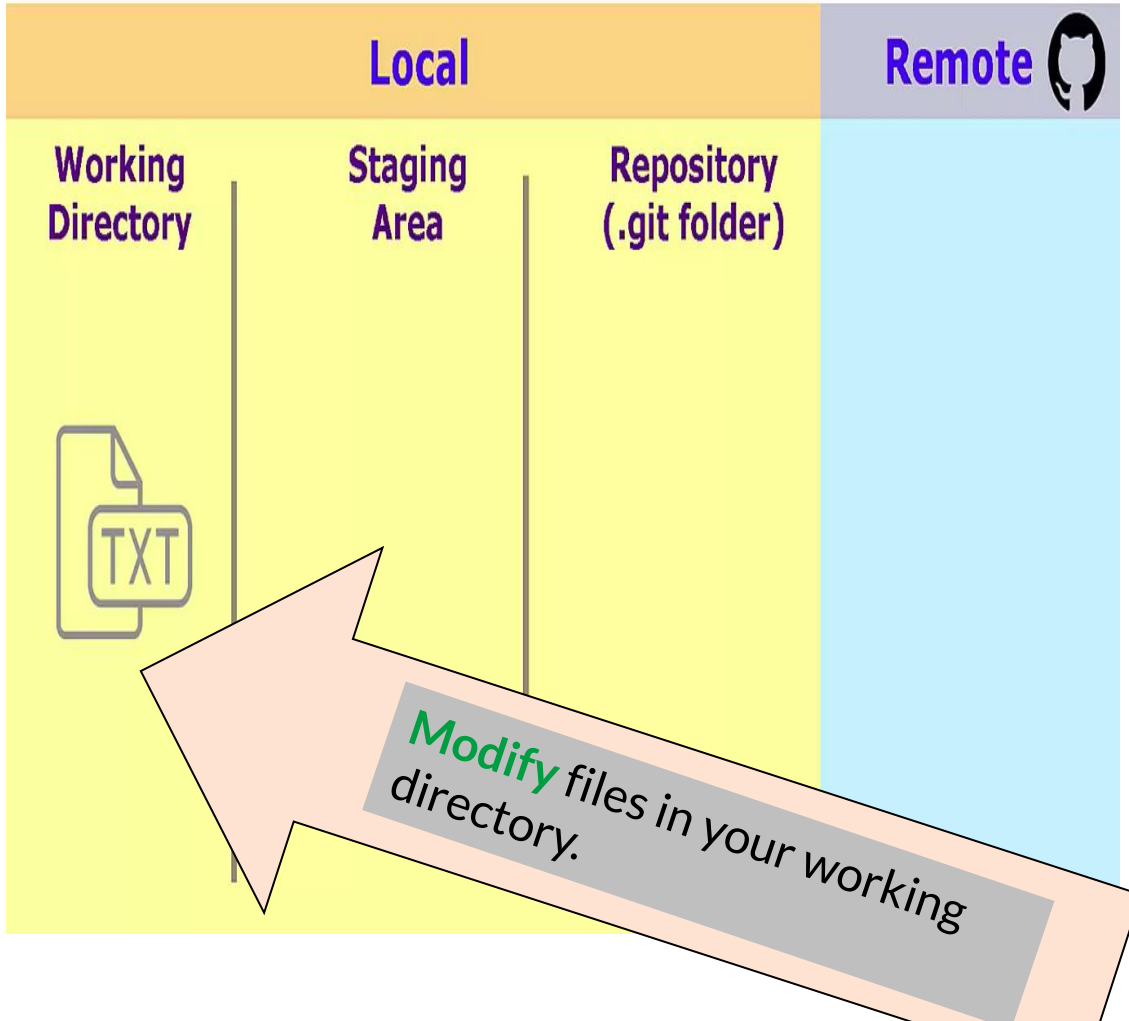
are pulled out of the compressed database in the Git directory and placed on disk

to edit and use.

Staging area – It is a simple file, generally present in your Git directory, that stores

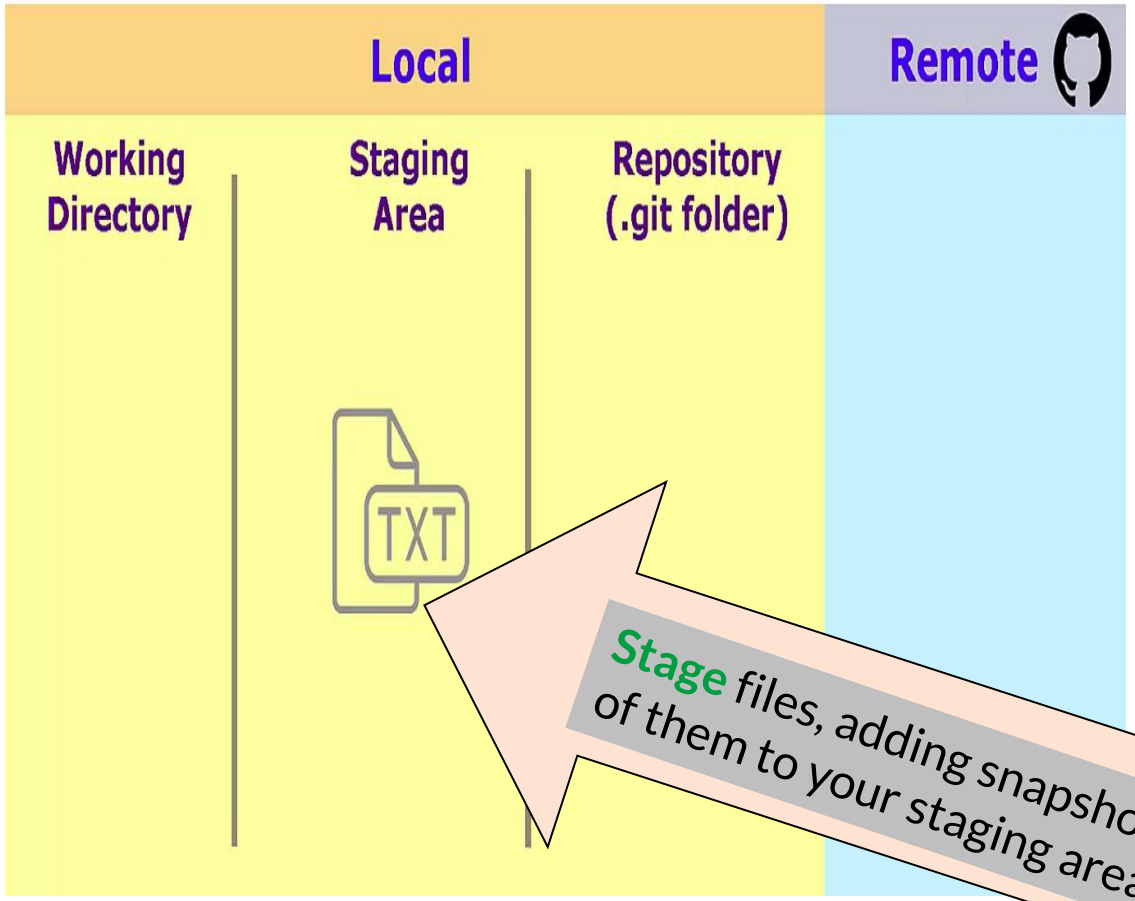
information about the next commit.

Basic Git Workflow Life Cycle



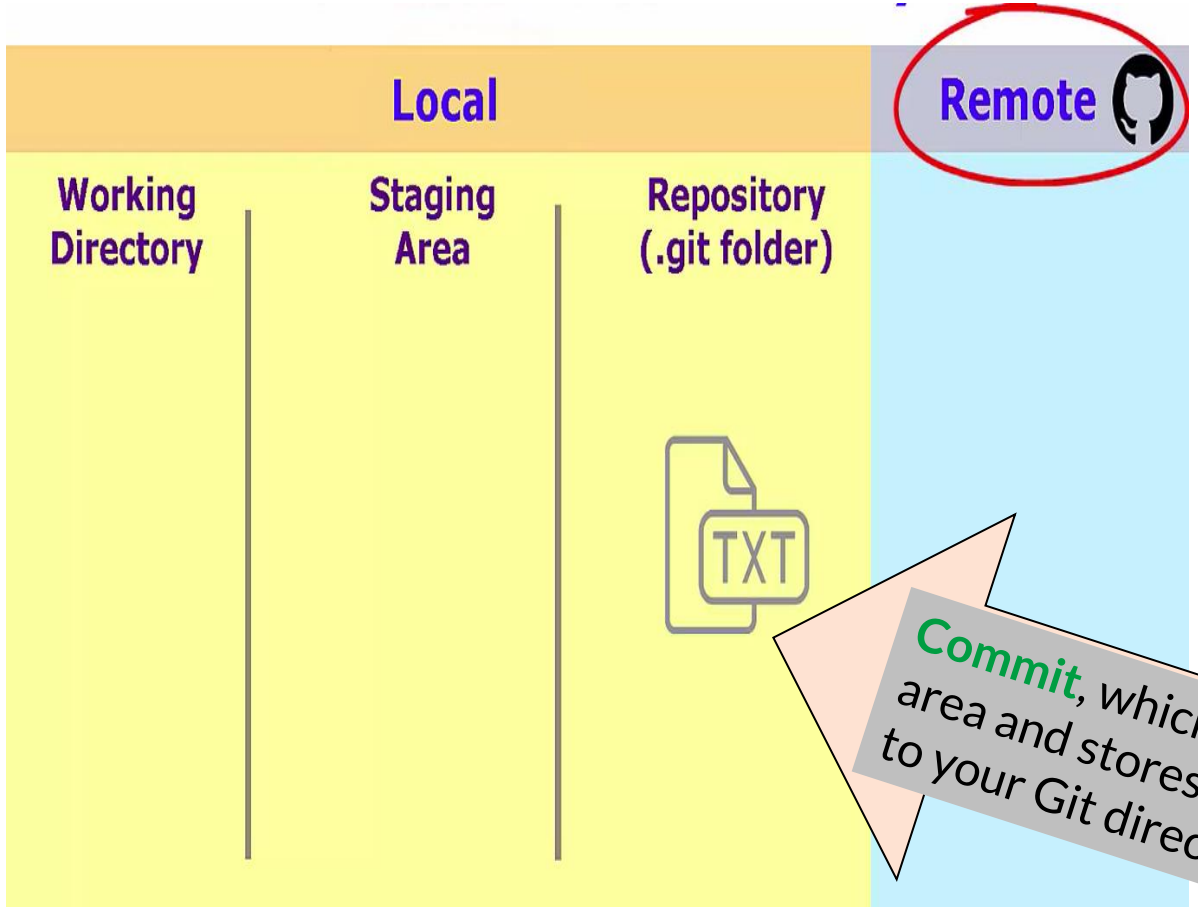

```
37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$ echo "Test Git Lifecycle workflow" >> Test.txt

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$ cat Test.txt
Test Git Lifecycle workflow

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        Test.txt

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

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$
```

Basic Git Workflow Life Cycle (Cont..)




```
37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$ git add Test.txt

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   Test.txt


37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$
```

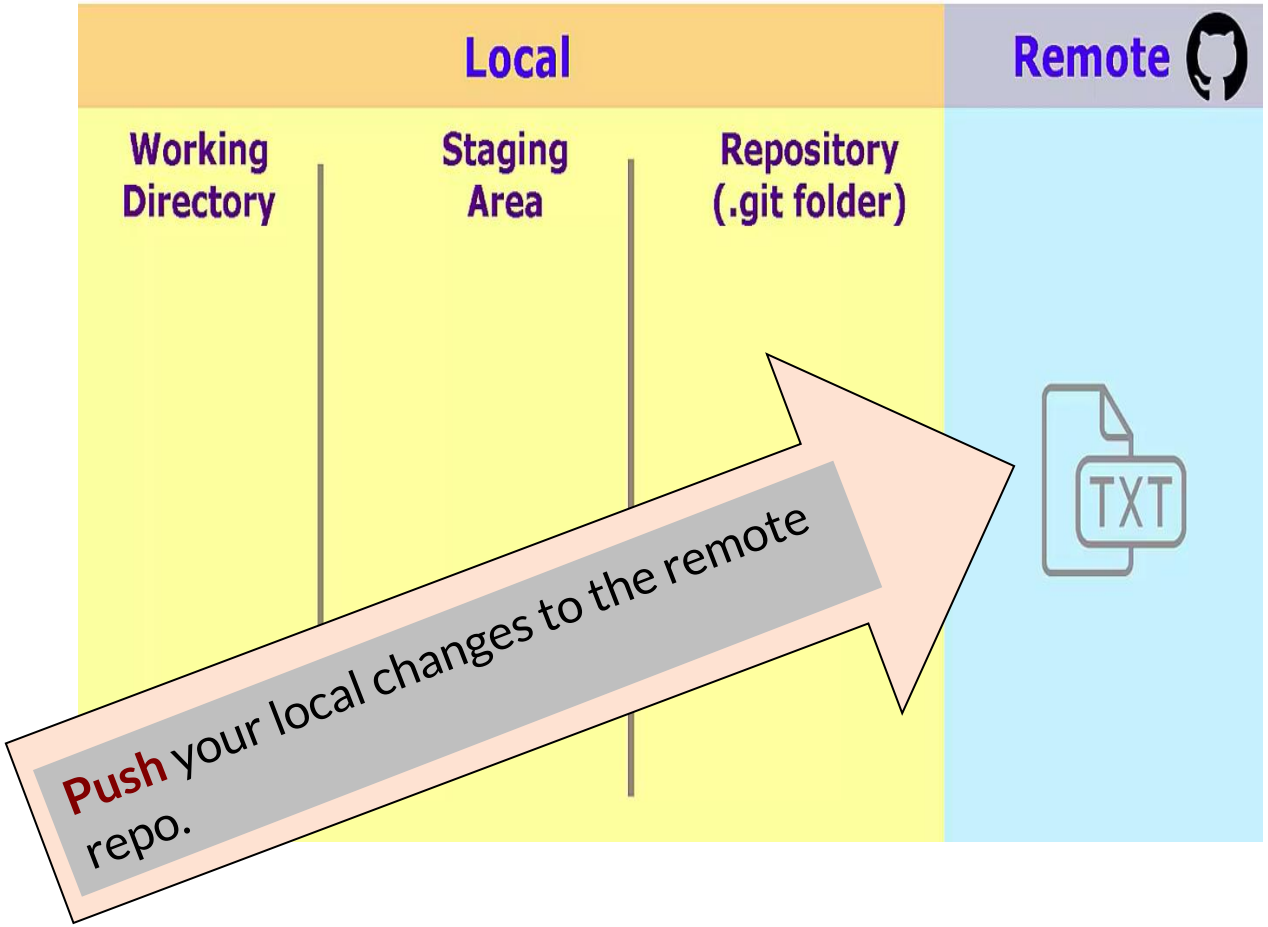
Basic Git Workflow Life Cycle (Cont..)




```
37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$ git commit -m "Adding test file"
[master d7fec3d] Adding test file
 1 file changed, 1 insertion(+)
 create mode 100644 Test.txt

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working tree clean

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$
```

Commit, which takes the files in the staging area and stores that snapshot permanently to your Git directory.

Basic Git Workflow Life Cycle (Cont..)




```
37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working tree clean

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$ git push origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 336 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:HexaInnovLab/sample.git
   a325890..d7fec3d  master -> master

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$
```

```
(master) github-demo $ git push origin master
```

origin refers to the GitHub copy of our repository

Initial GIT Configuration

- Set the name and email for Git to use when you commit:
  - **git config --global user.name "Bugs Bunny”**
  - **git config --global user.email bugs@gmail.com**

- **You can call git config –list to verify these are set.**

Creating a GIT Repo

- Two common scenarios: (only do one of these)

To create a new local Git repo in your current directory:

git init

- This will create a **.git** directory in your current directory.
- Then you can commit files in that directory into the repo.

– git add filename
– git commit –m "commit message“

To clone a remote repo to your current directory:

git clone url localDirectoryName

- This will create the given local directory, containing a working copy of the files from the repo, and a **.git** directory

(used to hold the staging area and your actual local repo)

GIT Commands

command	description
git clone ***url [dir]***	copy a Git repository so you can add to it
git add ***file***	adds file contents to the staging area
git commit	records a snapshot of the staging area
git diff	shows diff of what is staged and what is modified but unstaged
git help *[**command**]*	get help info about a particular command
git pull	fetch from a remote repo and try to merge into the current branch
git push	push your new branches and data to a remote repository
git fetch	imports commits from a remote repository into your local repo.
git revert	is for undoing shared public changes
git reset	is for undoing local private changes.

GIT Commit

Used to capture a snapshot of the project's currently staged changes.

git commit

- Commit the staged snapshot.

[This will launch a text editor prompting you for a commit message. After you've entered a message, save the file and close the editor to create the actual commit.]

git commit –a

- Commit a snapshot of all changes in the working directory

git commit –m "commit message"

- A shortcut command that immediately creates a commit with a passed commit message.

Viewing History

```
37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/sample (master)
$ echo "Hello" >> myfile.txt

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/sample (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        myfile.txt

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

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/sample (master)
$ git add myfile.txt
warning: LF will be replaced by CRLF in myfile.txt.
The file will have its original line endings in your working directory.

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/sample (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   myfile.txt


37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/sample (master)
$ git commit -m "myfile.txt added"
[master 4c40886] myfile.txt added
 1 file changed, 1 insertion(+)
 create mode 100644 myfile.txt
```

Git log:

Used to see the history of commits.

```
37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/sample (master)
$ git log
commit 4c40886c80ee7e904babdb376cf7d867139ce3ed (HEAD -> master)
Author: jamunarani <jamunaranik@hexaware.com>
Date:   Tue Mar 20 12:23:13 2018 +0530

    myfile.txt added
```

Viewing History (Cont..)

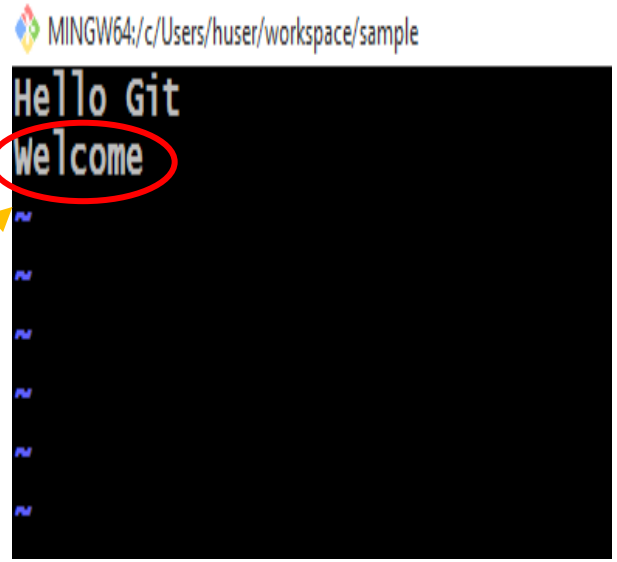
```
37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/sample (master)
$ vi myfile.txt

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/sample (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
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:   myfile.txt

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

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/sample (master)
$ git add myfile.txt
warning: LF will be replaced by CRLF in myfile.txt.
The file will have its original line endings in your working directory.

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/sample (master)
$ git commit -m "updated"
[master 6c9876d] updated
 1 file changed, 1 insertion(+)
```

```
37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/sample (master)
$ git log
commit 6c9876daf55689802f12be0f9727a7dc3ac8b681 (HEAD -> master)
Author: jamunarani <jamunaranik@hexaware.com>
Date:   Tue Mar 20 12:26:50 2018 +0530

    updated

commit 4c40886c80ee7e904babdb376cf7d867139ce3ed
Author: jamunarani <jamunaranik@hexaware.com>
Date:   Tue Mar 20 12:23:13 2018 +0530

    myfile.txt added
```

Git diff

- To see what is modified but unstaged:
  - **git diff**

```
370560@HVLAP-07 MINGW64 /c/Users/huser/workspace (master)
$ echo "Hello git" >> Test.txt

370560@HVLAP-07 MINGW64 /c/Users/huser/workspace (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        Test.txt

370560@HVLAP-07 MINGW64 /c/Users/huser/workspace (master)
$ git add Test.txt
warning: LF will be replaced by CRLF in Test.txt.
The file will have its original line endings in your working directory.

370560@HVLAP-07 MINGW64 /c/Users/huser/workspace (master)
$ git diff

370560@HVLAP-07 MINGW64 /c/Users/huser/workspace (master)
$ vi Test.txt

370560@HVLAP-07 MINGW64 /c/Users/huser/workspace (master)
$ git diff
warning: LF will be replaced by CRLF in Test.txt.
The file will have its original line endings in your working directory.
diff --git a/Test.txt b/Test.txt
index 0dec223..63f8506 100644
--- a/Test.txt
+++ b/Test.txt
@@ -1 +1,2 @@
 Hello git
+welcome
```



Git diffs cont..

- To see a list of staged changes:
  - **git diff --cached**

```
37056@HVLAP-07 MINGW64 /c/Users/huser/workspace (master)
$ git add Test.txt
warning: LF will be replaced by CRLF in Test.txt.
The file will have its original line endings in your working directory.

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace (master)
$ git diff

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace (master)
$ git diff --cached
diff --git a/Test.txt b/Test.txt
new file mode 100644
index 0000000..63f8506
--- /dev/null
+++ b/Test.txt
@@ -0,0 +1,2 @@
+Hello git
+welcome
```

Undo changes on a file

- To undo changes on a file before you have committed it:

  – git checkout -- filename
  (undo the uncommitted changes)

  - git checkout --.
  (*dot* undo all the uncommitted changes)

```
37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$ vi start.txt

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/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:   start.txt

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

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$ git checkout start.txt

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$
```

  – All these commands are acting on your local version of repo.

Git Reset command

- Git reset allows us to move commits from history back into working or staging area.
- It can also be used to throw the changes away.
- **- git reset HEAD – filename**

```
37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$ git add start.txt

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   start.txt


37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$ git reset HEAD start.txt
Unstaged changes after reset:
M       start.txt

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/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:   start.txt

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

37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$
```

Three options of Git Reset

- Git reset --soft
- Git reset --mixed
- Git reset --hard

Working Directory	Staging Area	.git directory (Repository)
		git reset --soft
		git reset --mixed
		git reset --hard

Git Reset

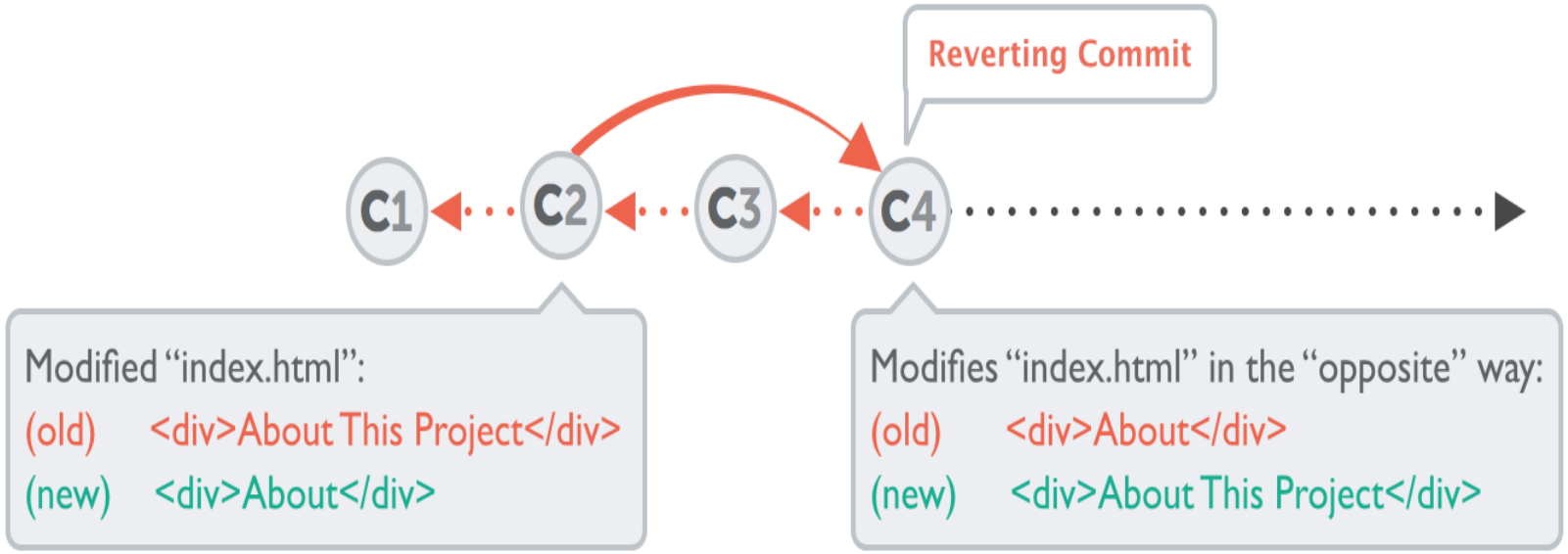
git log –oneline -> Shows the list of commits in the project.

```
37056@LTCH-5CD91121FN MINGW64 ~/workspace/MLP240 (newbranch)
$ git log --oneline
2c5b5d0 (HEAD -> newbranch) Third change
b704c56 second change
321c6f5 first change

37056@LTCH-5CD91121FN MINGW64 ~/workspace/MLP240 (newbranch)
$ git reset --mixed b704c56
Unstaged changes after reset:
M       myfile1.txt

37056@LTCH-5CD91121FN MINGW64 ~/workspace/MLP240 (newbranch)
$ git status
On branch newbranch
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   myfile1.txt

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

Committed changes are moved from local repo to working directory.

Git Revert

- Git Revert is to create a new commit which reverts a previous commit.
- Its reverting a commit that has been pushed to the remote.

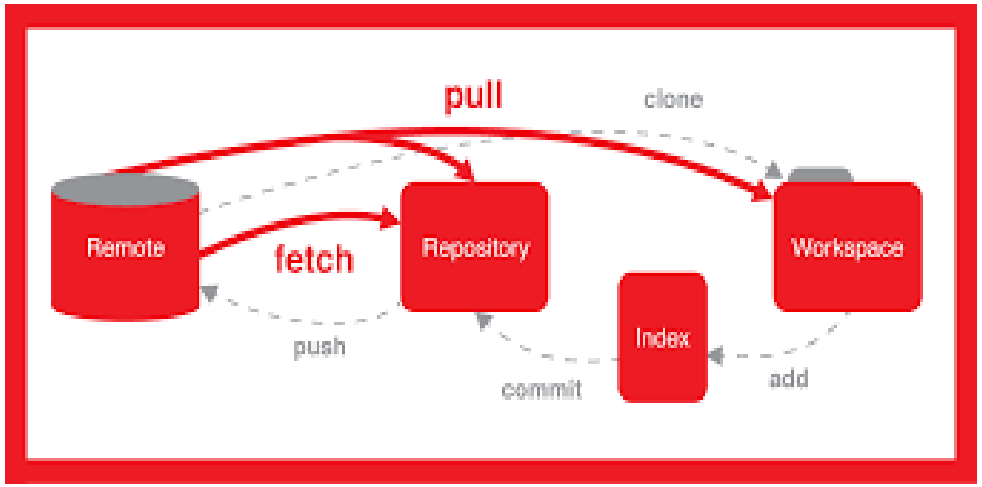


Git Fetch

- Git Fetch downloads new data from a remote repository.
- It retrieves any commits, branches and files from a remote repository, along with any other objects.

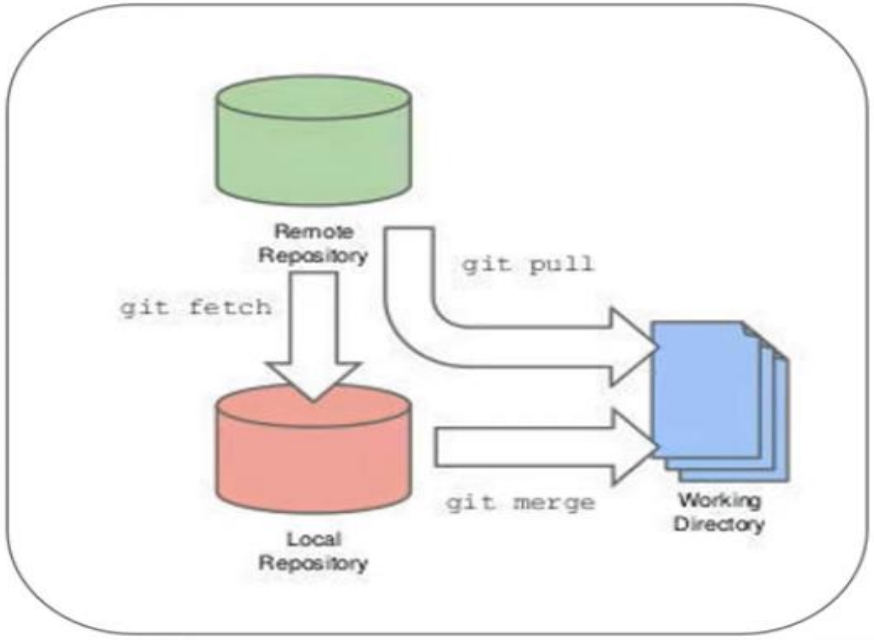
- **git fetch**

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/Git-Example (master)
$ git fetch origin

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/Git-Example (master)
$ git fetch origin
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/ImDwivedi1/Git-Example
 * [new branch]      test2      -> origin/test2

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/Git-Example (master)
$
```



Git Fetch Vs Git Pull

Git Fetch	Git Pull
Git fetch is used to retrieve the latest meta-data info from the original.	Git pull is used to bring those changes from the remote repository. **git pull=git fetch + merge**
Git Fetch never manipulates or spoils data.	Git Pull downloads the data and integrates it with the current working file.
It protects your code from merge conflict.	In git pull, there are more chances to create the merge conflict.



Branching

Branching

Git uses branching heavily to switch between multiple tasks.

Branch Creation:

- To create a new local branch:
  - **– git branch name**
- To list all local branches: (* = current branch)
  - **– git branch**
- To list all remote branches:
  - **- git branch -r**
- To list all local and remote branches:
  - **- git branch -a**
- To switch to a given local branch:
  - **– git checkout branchname**
- Creates and checkouts to a branch called <name>
  - **- git checkout -b <name>**

Branching

Cont..

Branch Deletion:

- To delete a local branch:
  - **– git branch –d <name>**
  - **- git branch –D <name>**
    - **Use –D for force deletion even if it hasn't been pushed or merged yet**
- To delete a remote branch:
  - **– git push <remote> ––delete <branch>**
  - **- git push <remote> :<branch> [Shorter command]**

For example:

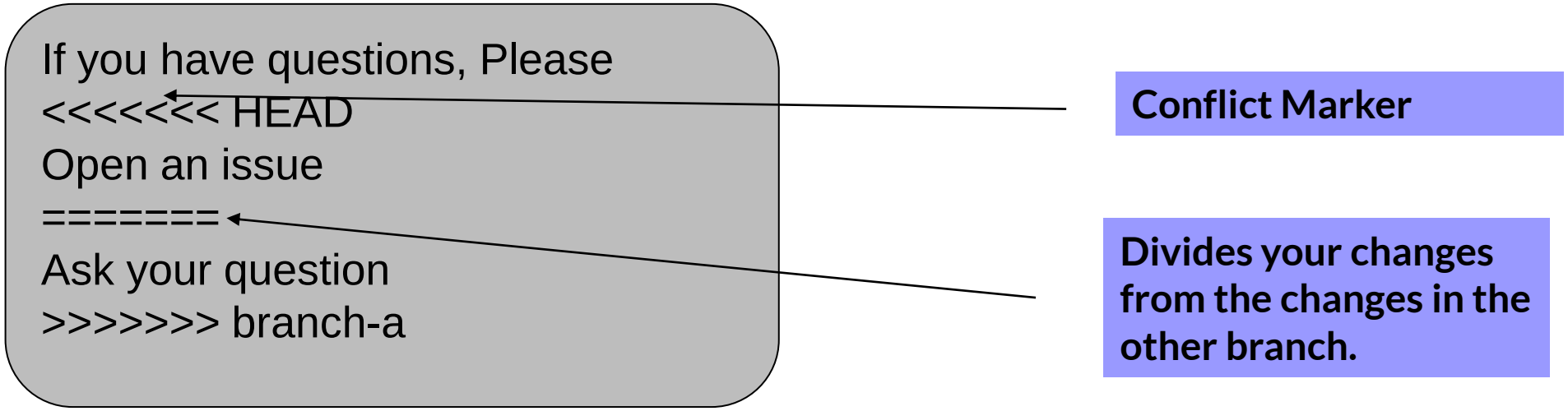
- **git push origin ––delete branchname**
- **git push origin :branchname**

Merge Conflicts

- Merge conflicts may occur if competing changes are made to the same line of a file or when a file is deleted that another person is attempting to edit.
- The conflicting file will contain <<< and >>> sections to indicate where Git was unable to resolve a conflict:
- Find all such sections, and edit them to the proper state (whichever of the two versions is newer / better / more correct).

Merge Conflicts

- For Example:
  - If one person wrote "open an issue" in the base or HEAD branch
  - And another person wrote "ask your question in the compare branch
  - Decide if you want to
    - Keep only your branch's changes,
    - or other branch's changes,
    - or make a brand-new change, which may incorporate changes from both branches.
  - Deletes the conflict marker <<<<<<< , ======= , >>>>>>> and make the changes.

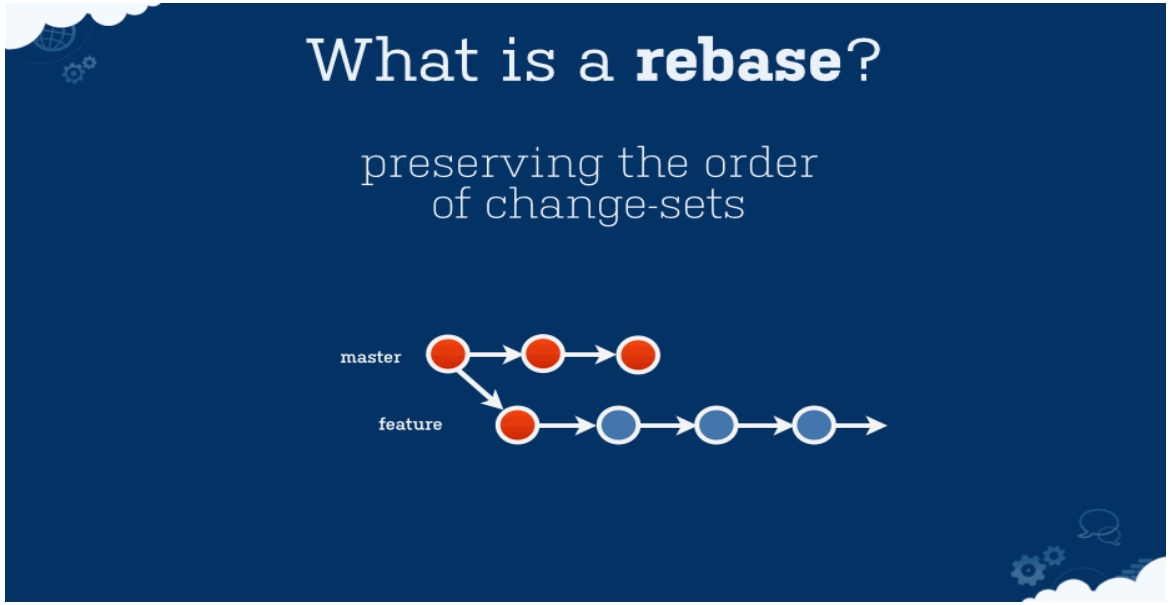


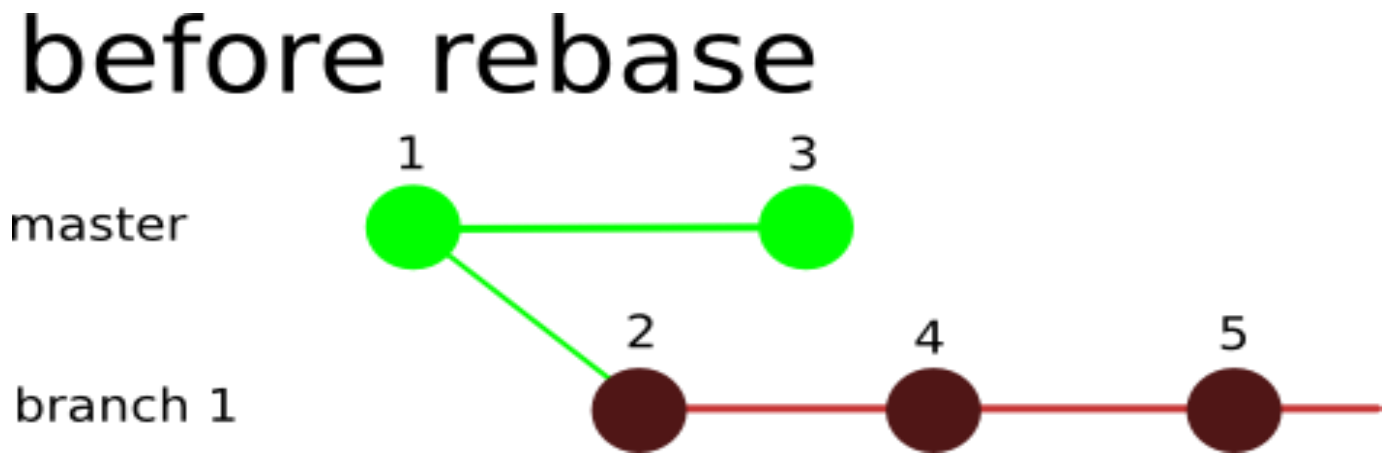
Rebase

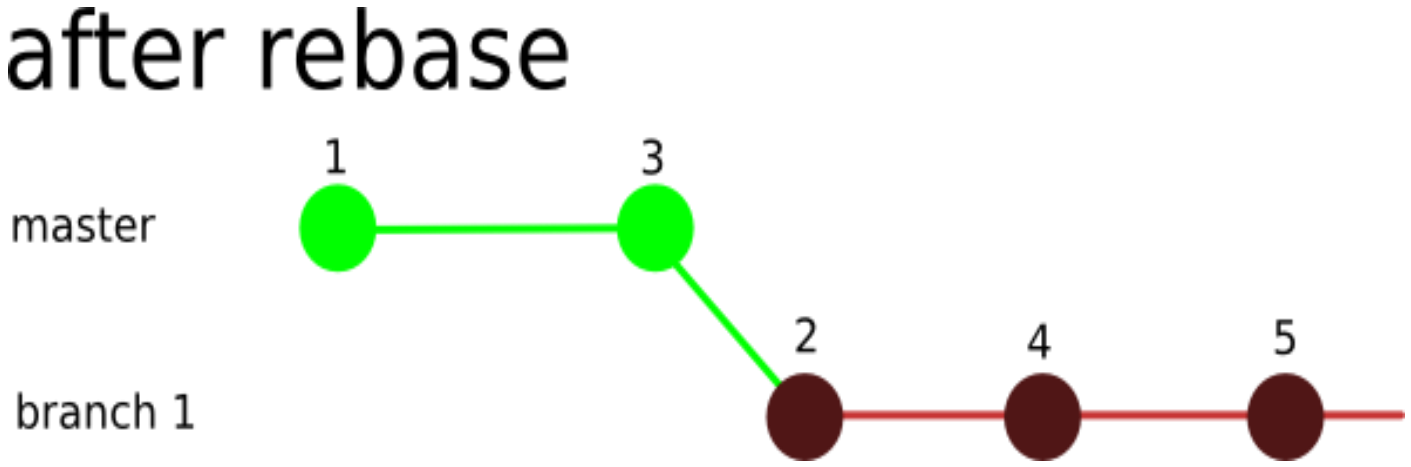
- Changes the "starting point" of a branch
- Can be used in feature branches when the "development branch" changes drastically and it affects the feature branches
- Usage:

  $ git rebase <branch>

  sets the starting point of the current branch to <branch>




before rebase
1
3
master
2
4
5
branch 1

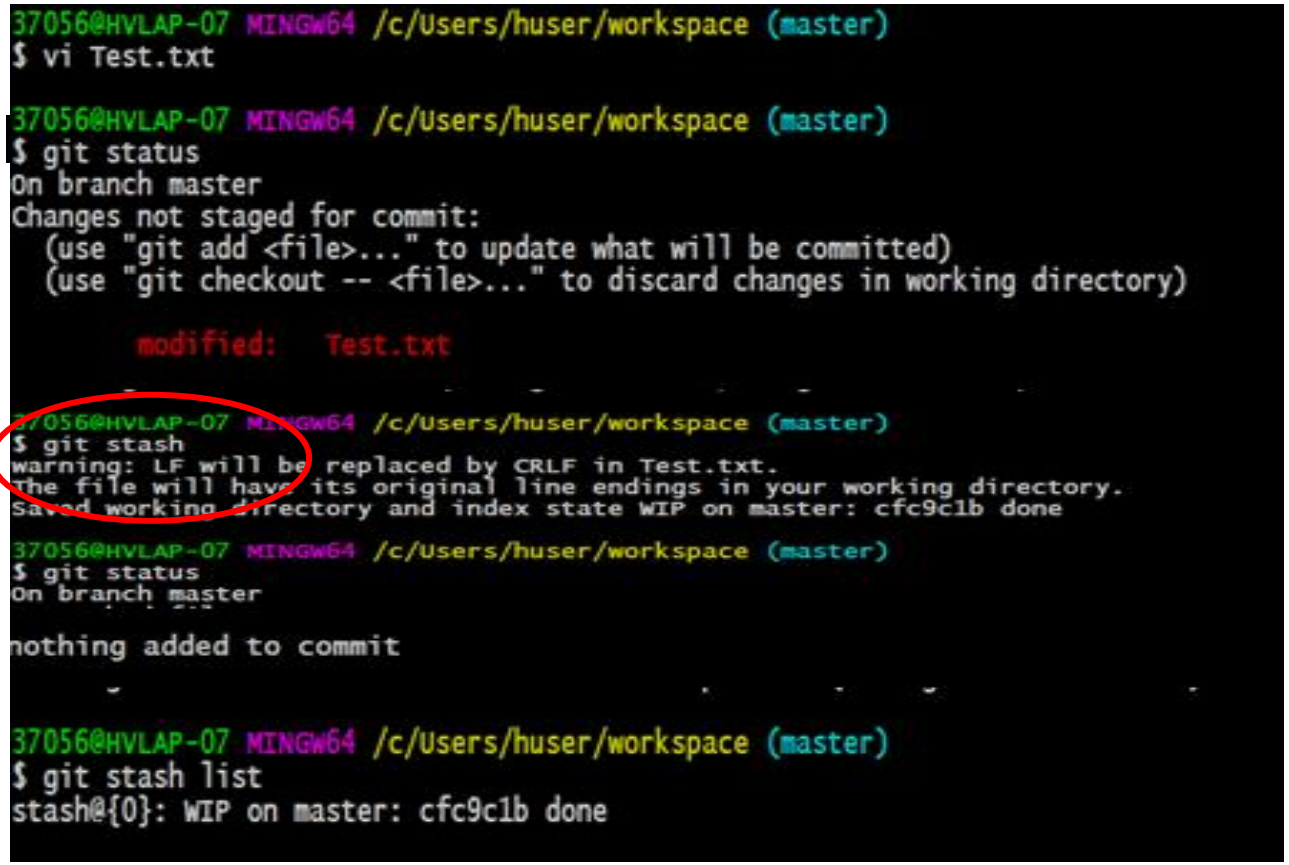

after rebase
1
3
master
2
4
5
branch 1

Interaction w/ remote repo

- **Push** your local changes to the remote repo.
- **Pull** from remote repo to get most recent changes.
  - (fix conflicts if necessary, add/commit them to your local repo)
- To fetch the most recent updates from the remote repo into your local repo, and put them into your working directory:
  - **git pull origin master**
- To put your changes from your local repo in the remote repo:
  - **git push origin master**

Stashing commits

- Stashing commits can be useful when you have made some changes in your working directory, but want to return to a ”clean working directory”
  Like when you need to do a quick fix on something entirely else, but your hacking is not complete
- Usage
  Stashing your commits since last pu
  $ git stash
  Listing all stashed changes
  $ git stash list
  Recovering changes from stash
  $ git stash pop/apply
  Cleaning up your stash
  $ git stash drop



Git hosting services

- **GitHub:**

  GitHub.com is a site for online storage of Git repositories.

  – You can create a **remote repo** there and push code to it.

  – Many open-source projects use it, such as the Linux kernel.

  – You can get free space for open-source projects,
  or you can pay for private projects.

  - Free private repos for educational use: github.com/edu

- Other hosting services are
  - Bitbucket
  - GitLab
  - Google Cloud Source Repositories
  - Google Drive [for personal projects and small team collaboration]

Git hosting services – AWS CodeCommit

- It is a version control service
- Its fully managed by Amazon
- It offers high scalability, security and helps to manage source control service.
- Eliminates the administrative overhead from the user end of managing their own software and hardware.
- Provides highly available service, which is durable
- User's code is stored securely, since the Codecommit repositories are encrypted.
- Software development team members can work collaboratively on the code.
- Codecommit comes with support of Git command, and it can be used with AWS CLI and API as well.
- Codecommit can handle repos that have large files or branches that have large sizes and a long revision history.

Practice Exercise 1:

- Create a file **<your name>.txt** in the local repo. **e.g. John.txt**
- Add your unique talent in the **<your name>.txt** file. **e.g. public speaking**
- Do all the life cycle commands and merge your changes in the remote repo.

Practice Exercise 2:

- Create a new branch with the branch name ‘TeamDetails’
- Add myteams.txt in the branch.
- Each one of you in the team add your name in the myteams.txt file.
- Do all life cycle commands, rebase it, resolve the conflicts and merge your changes in the remote repo.

For e.g.

myteams.txt
- Sayan Ghosh
- Shiva sujan
- Rishav
- Ruchi

Practice Exercise 3:

- **Story Creation: [Decide a story among the teams]**
- **In the local repo,**
- **Team member 1 :**
    1. create a branch ‘character’
    2. Add <storyname>.txt file in the ‘character’ branch
    3. list the characters and give an intro about the characters
- **Team member 2 :**
    1. create a branch ‘setting’
    2. Add <storyname>.txt file in the ‘setting’ branch
    3. Describe the setting of the story in <storyname>.txt file
- **Team member 3 :**
    1. create a branch ‘plot’
    2. Add <storyname>.txt file in the ‘plot’ branch
    3. Write the plot of the story in <storyname>.txt file

Practice Exercise 3:

- **Team member 4:**
  1. create a branch ‘conflict’
  2. Add <storyname>.txt file in the ‘conflict’ branch
  3. Write the conflict of the story in <storyname>.txt file
- **Team member 5:**
  1. create a branch ‘theme’
  2. Add <storyname>.txt file in the ‘theme’ branch
  3. Write the theme of the story in <storyname>.txt file
  - **Perform all the life cycle commands, do rebase, resolve the conflicts, create the pull request one by one and merge the changes in the remote repo.**
  - **Once merged, delete the local and remote branches.**

Note: The remote repo should contain only one file <storyname>.txt, which has character, setting, plot, conflict & theme.

Question 1:

- Which of these terms best describes GIT?

a) Issue Tracking
b) Integerated Development Environment
c) Web-Based Repository Hosting Service
d) Distributed Version Control System

Ans : Distributed Version Control System

Question 2:

- Which of these Git clients commands creates a copy of the repository and a working directory in the client's workspace?

a) Import
b) Clone
c) Checkout
d) Merge

Ans: Clone

Question 3:

- Now, Imagine that you have a local repository, but other team members have pushed changes into the remote repository. What GIT operation would use to download those changes into your working copy?

a) Export
b) Commit
c) Checkout
d) Pull

Ans: Pull

Question 4:

- In Git which error would you get if you try to push master-branch changes to a remote repository and someone else pushed changes to that same branch while you were making your changes?

  a) Rejected
  b) 404
  c) 500
  d) Access denied

Ans : Rejected

Question 5:

- If you want to make radical changes to your team's project and don't want to impact the rest of the team, you should implement your changes in …

  a) The root
  b) a tag
  c) a branch
  d) The trunk

Ans : a branch

Thank you

Innovative Services

Passionate Employees

Delighted Customers