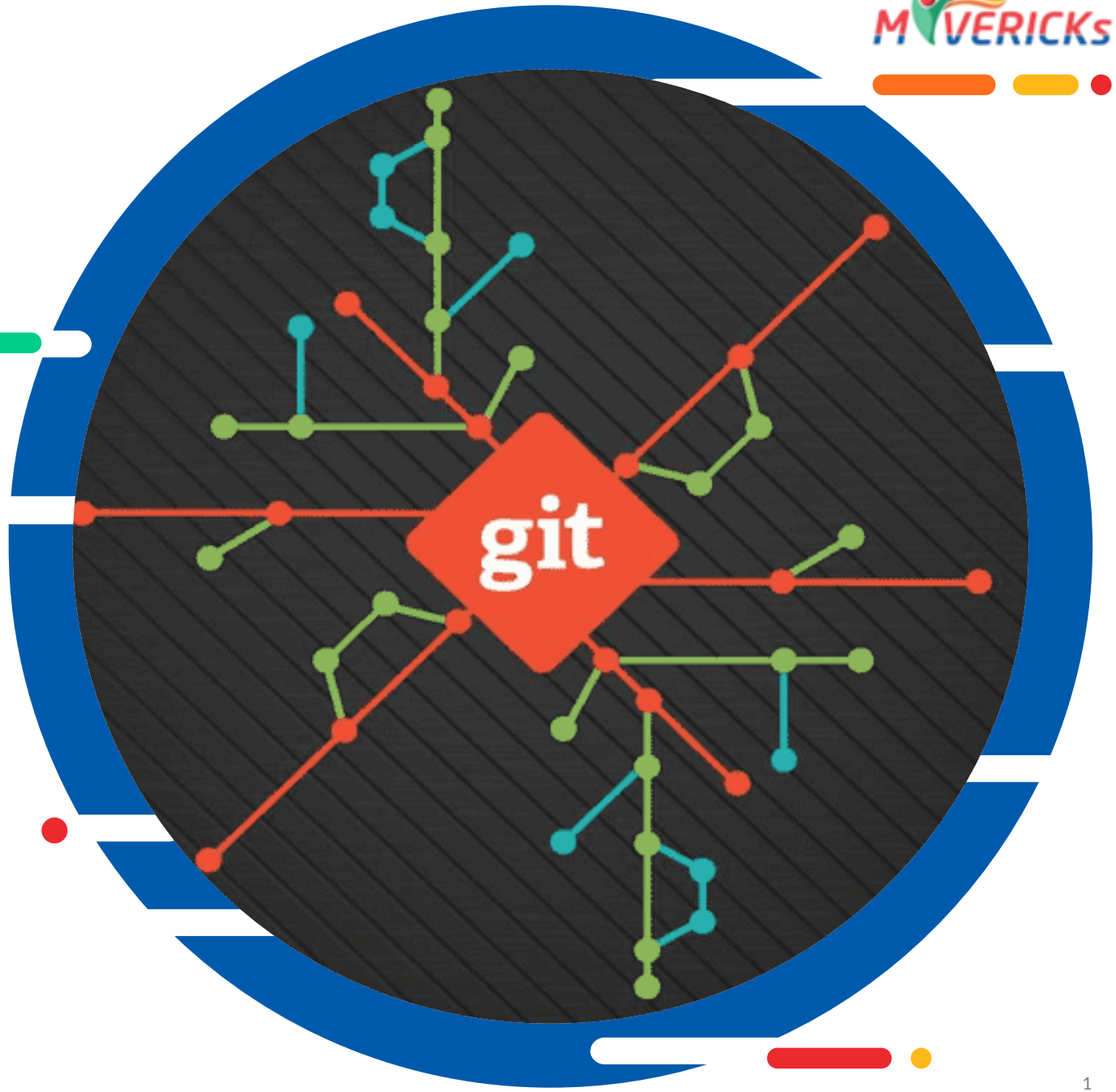
GIT



Objective

In this session, you will:
Understand Git version control system and its commands.


GAMIFICATION

Gamification

Objective:

To make the participants familiarize with Git through activity.

Activity on GIT

Session Plan

- ✓ About Git
- ✓ Version Control Systems
- ✓ Distributed VCS (GIT)
- ✓ Local Git areas
- ✓ Git workflow
- ✓ Creating Git Repo
- ✓ Git commands
- ✓ Git commit
- ✓ Undo changes on file
- ✓ Viewing history and diffs
- ✓ Branching
- ✓ Rebase
- ✓ Interaction with Remote Repo
- ✓ Stashing commits

About Git

- Created by **Linus Torvalds**, creator of Linux, in 2005
  - Came out of Linux development community
  - Designed to do version control on Linux kernel

- **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)

References

- https://www.atlassian.com/git/tutorials
- The standard one:
  http://git-scm.com
- Free on-line book: http://git-scm.com/book
- Git tutorial: http://schacon.github.com/git/gittutorial.html
- From StackExchange:
  http://stackoverflow.com/questions/315911/git-for-beginners-the-definitive-practical-guide#323764
- From github- https://help.github.com/
- At command line: (where verb = config, add, commit, etc.)
  - **git help verb**

Version Control Systems

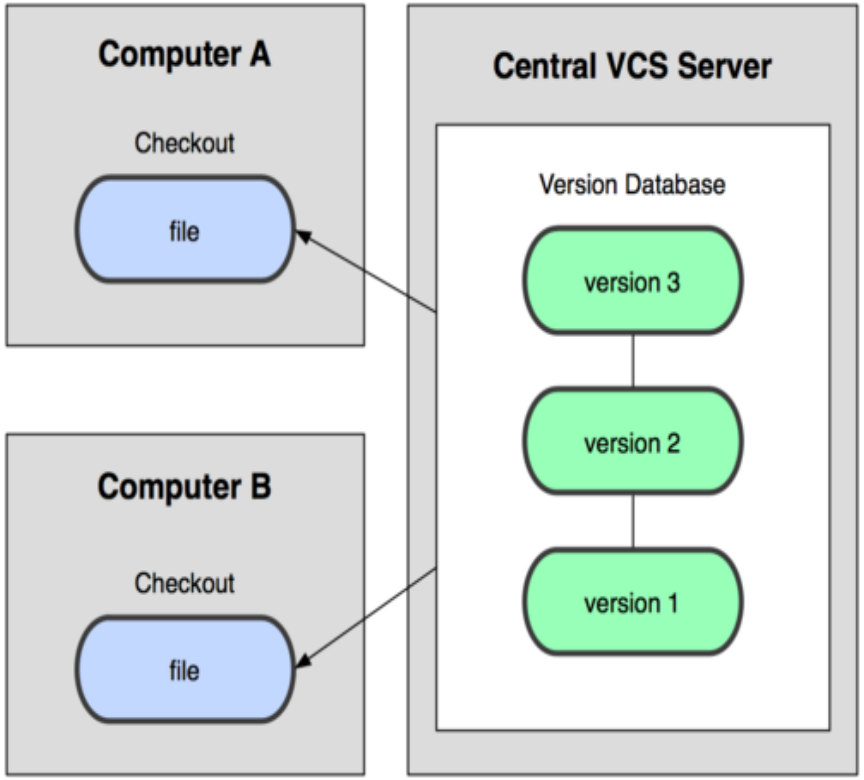
- Version control (or revision control, or source control) is all about managing multiple versions of documents, programs, web sites, etc.
  - Almost all “real” projects use some kind of version control
  - Essential for team projects, but also very useful for individual projects
- Some well-known version control systems are CVS, Subversion, Mercurial, and Git
  - CVS and Subversion use a “central” repository; users “check out” files, work on them, and “check them in”
  - Mercurial and Git treat all repositories as equal
- Distributed systems like Mercurial and Git are newer and are gradually replacing centralized systems like CVS and Subversion

Why Version Control?

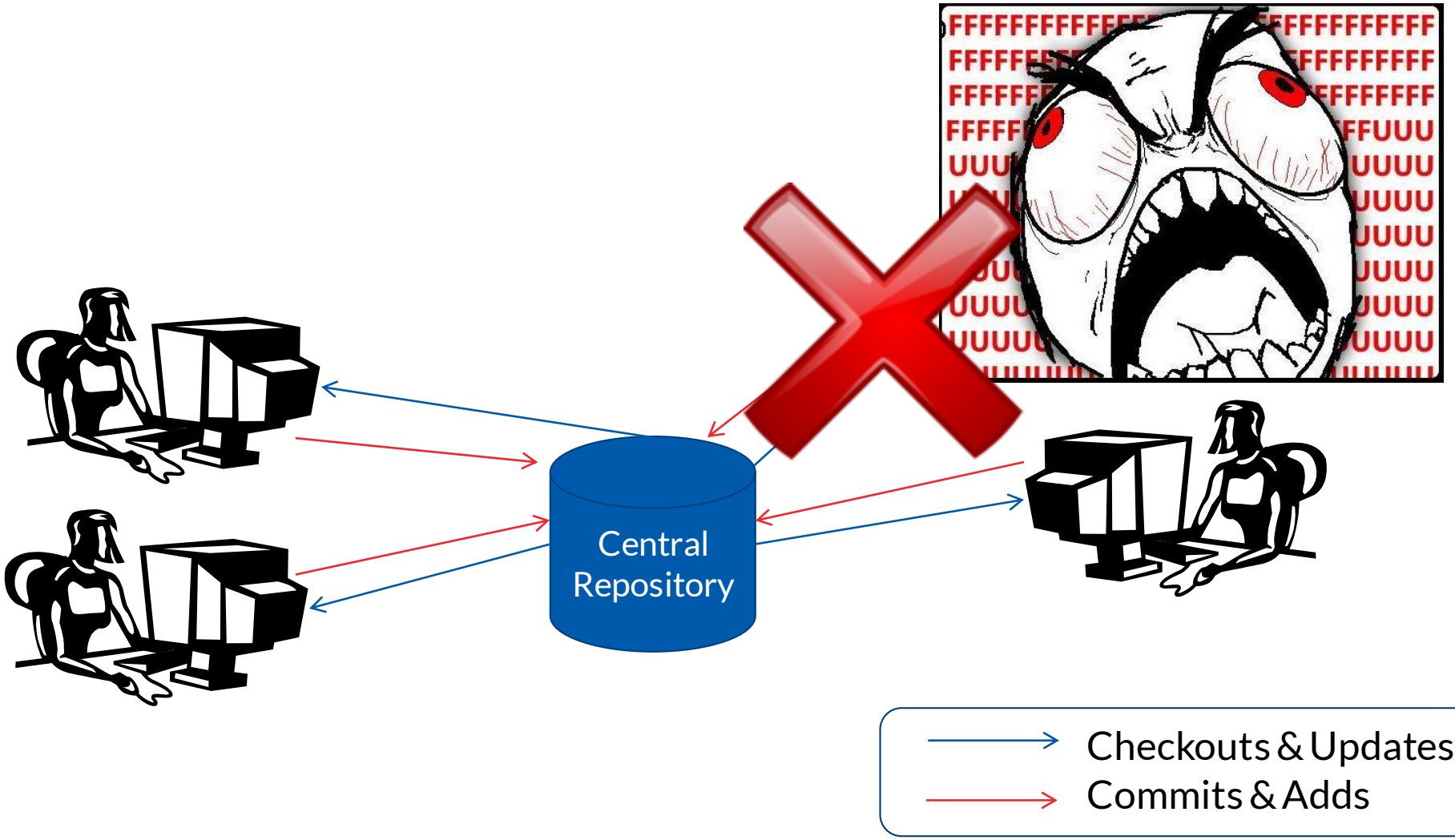
- **For working by yourself**:
  - Gives you a "time machine" for going back to earlier versions
  - Gives you great support for different versions (standalone, web app, etc.) of the same basic project
- **For working with others:**
  - Greatly simplifies concurrent work, merging changes

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



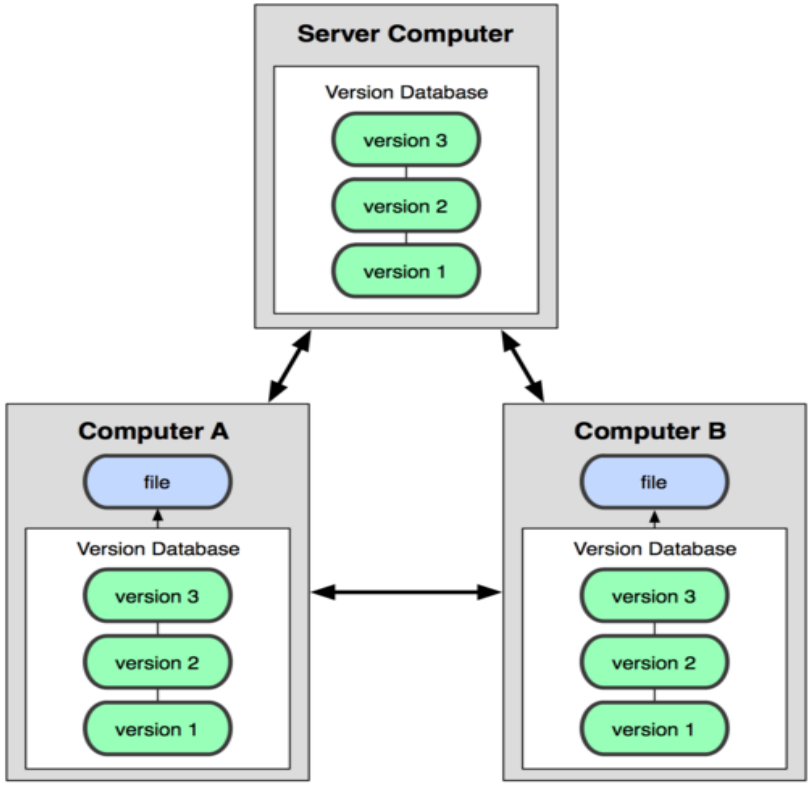
Traditional Centralized VC model (CVS, Subversion)



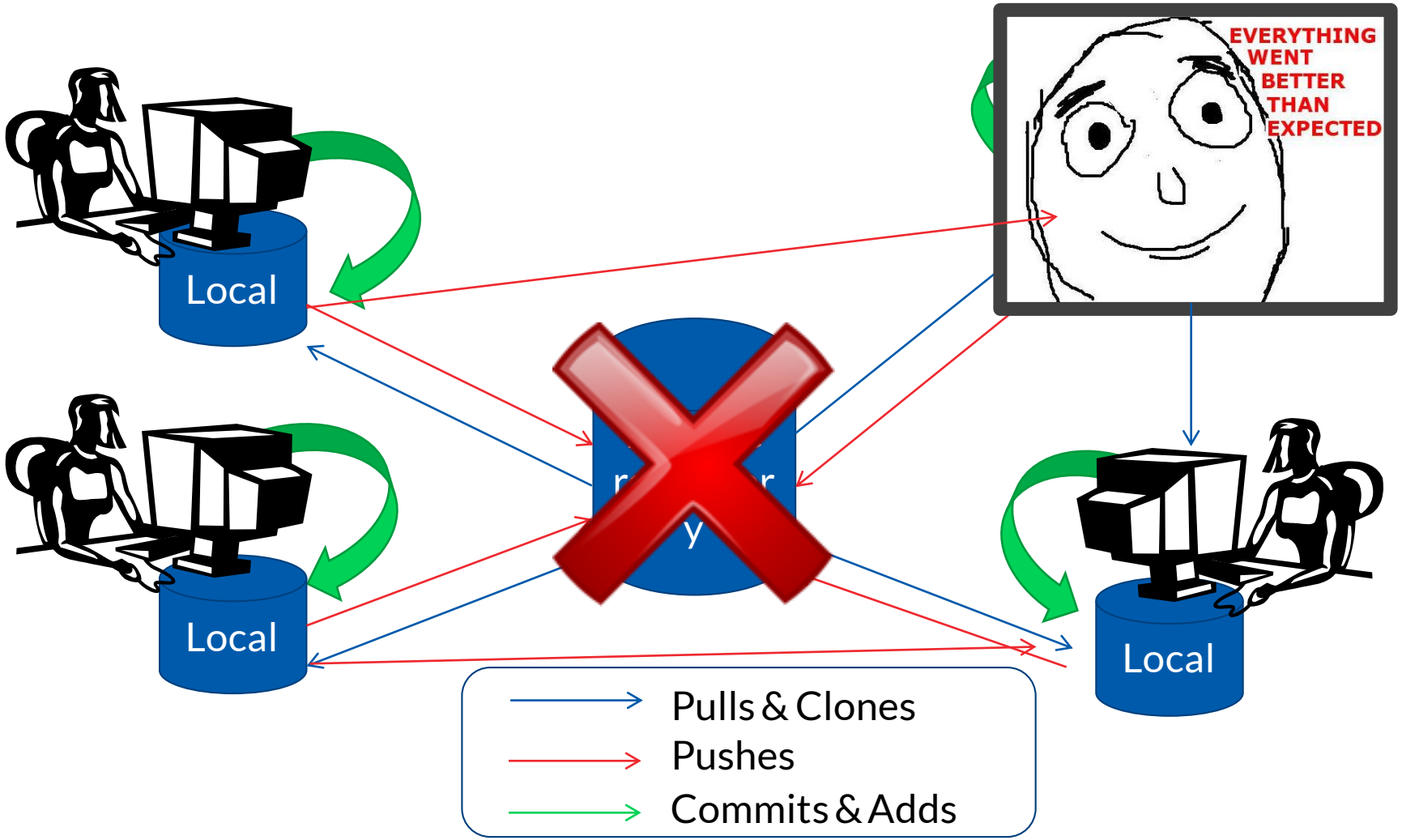
Distributed VCS (Git)

- In git, mercurial, etc., you don't "checkout" from a central repo
  – you "clone" it and "pull" changes from it
- Your local repo is a complete copy of everything on the remote server
  - - yours is "just as good" as theirs
- Many operations are local:
  - check in/out from local repo
  - commit changes to local repo
  - local repo keeps version history

- When you're ready, you can "push" changes back to server.
  -

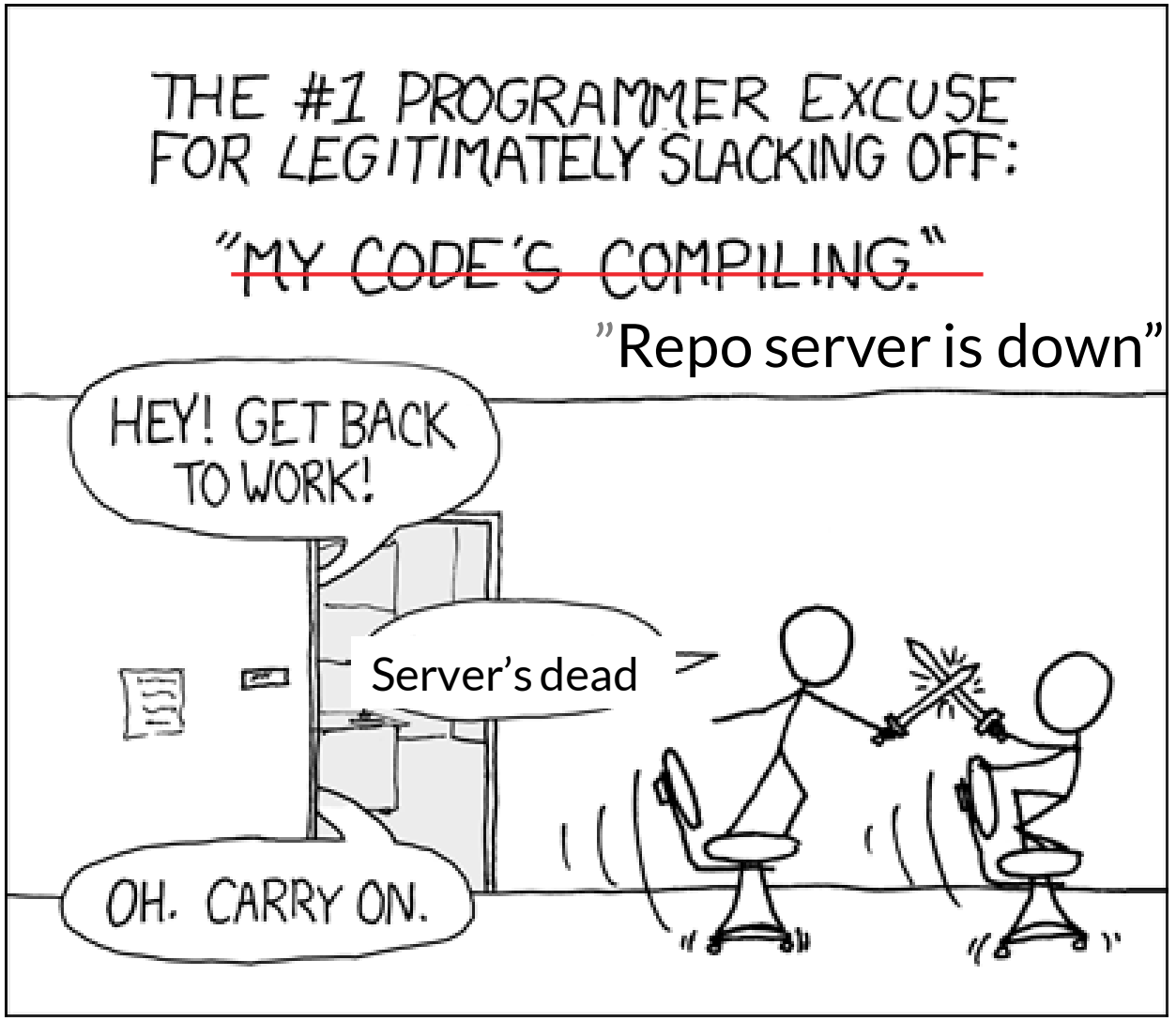


Distributed version control system (git)



Advantages

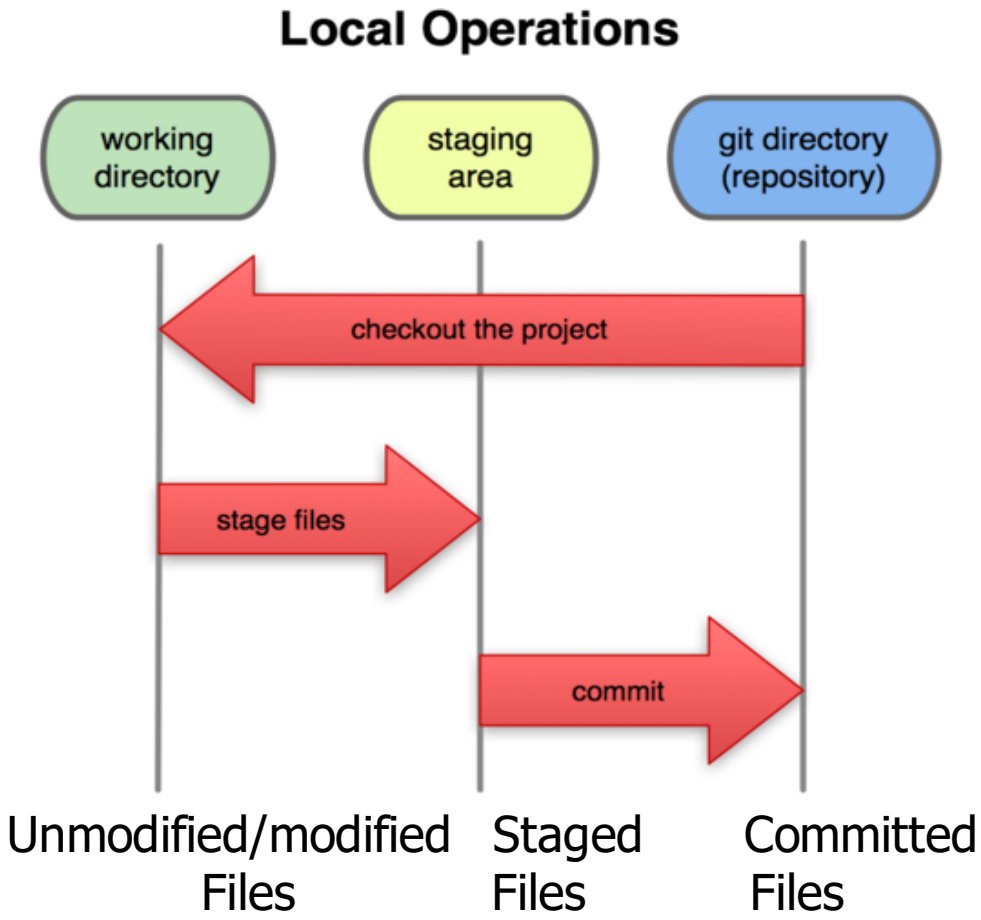
- ✓Changing branches can be done offline
- ✓Other revisions of a file can be obtained
- ✓Changes can be commited offline
- ✓Merging and Diff can be done offline



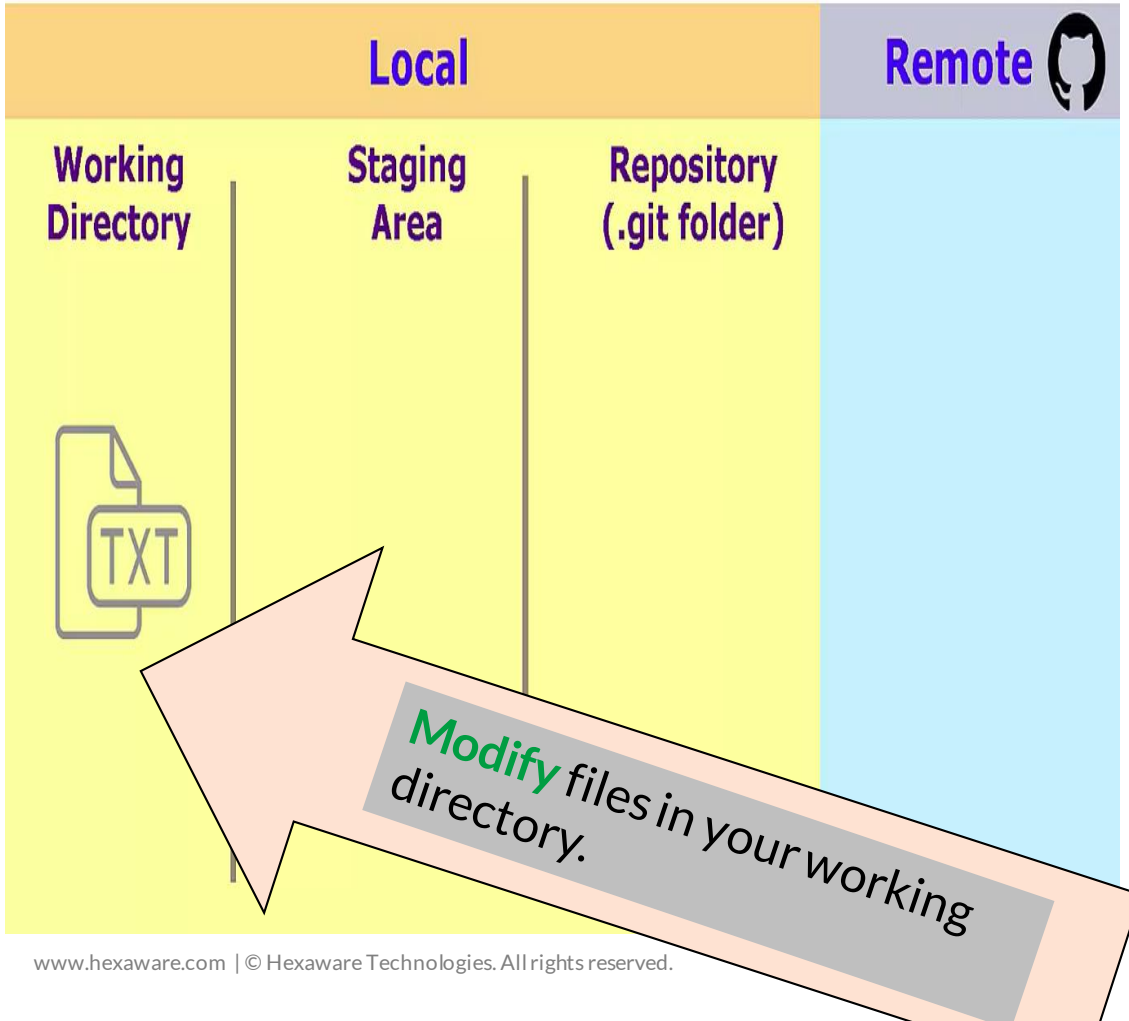
Local Git areas

In your local copy on git, files can be:

- In your local repo (committed)
- Checked out and modified, but not yet committed (working copy)
- Or, in-between, in a "staging" area
- Staged files are ready to be committed.
- A commit saves a snapshot of all staged state.

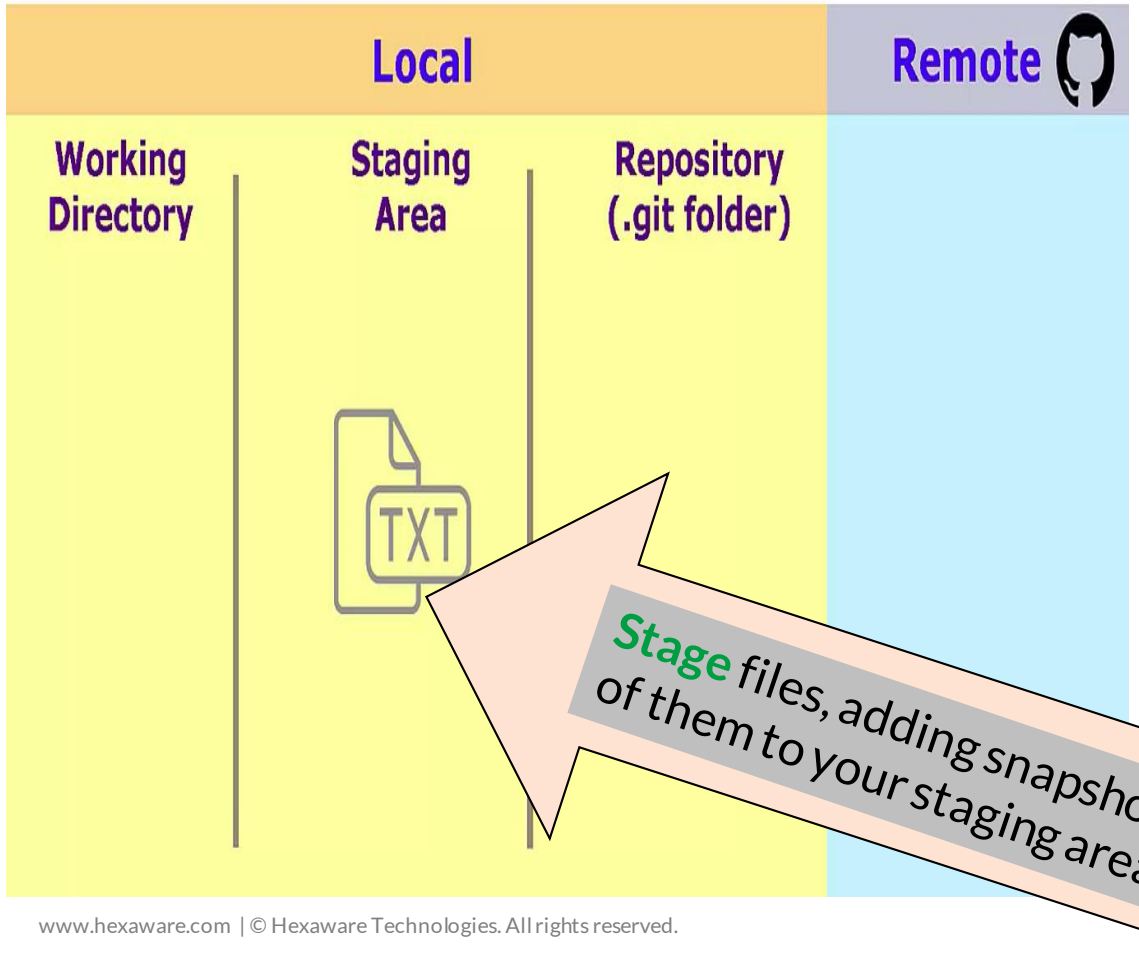


Basic Git Workflow Life Cycle



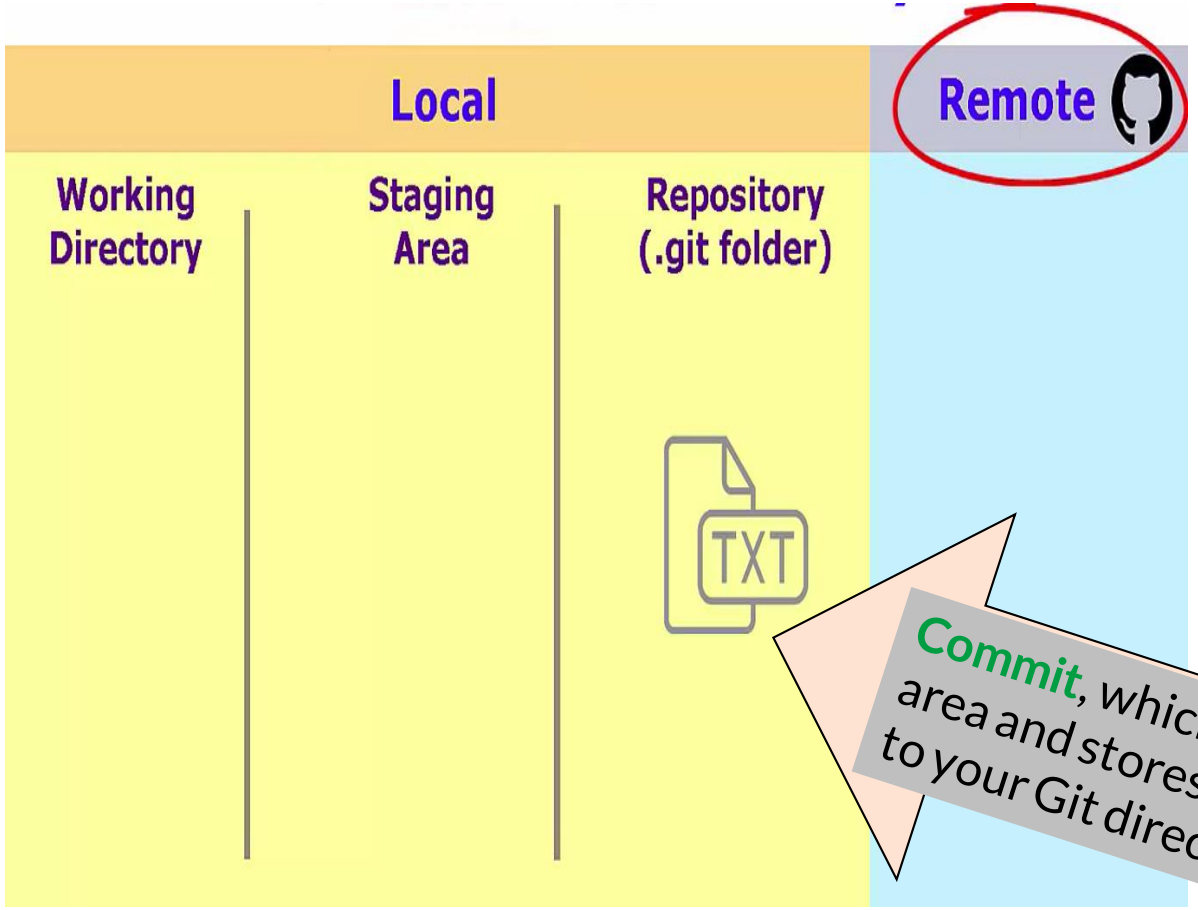

```
37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$ echo "Test Git Lifecycle workflow" >> Test.txt

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..)




```
370560@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$ git add Test.txt

370560@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


370560@HVLAP-07 MINGW64 /c/Users/huser/workspace/project/sample (master)
$
```

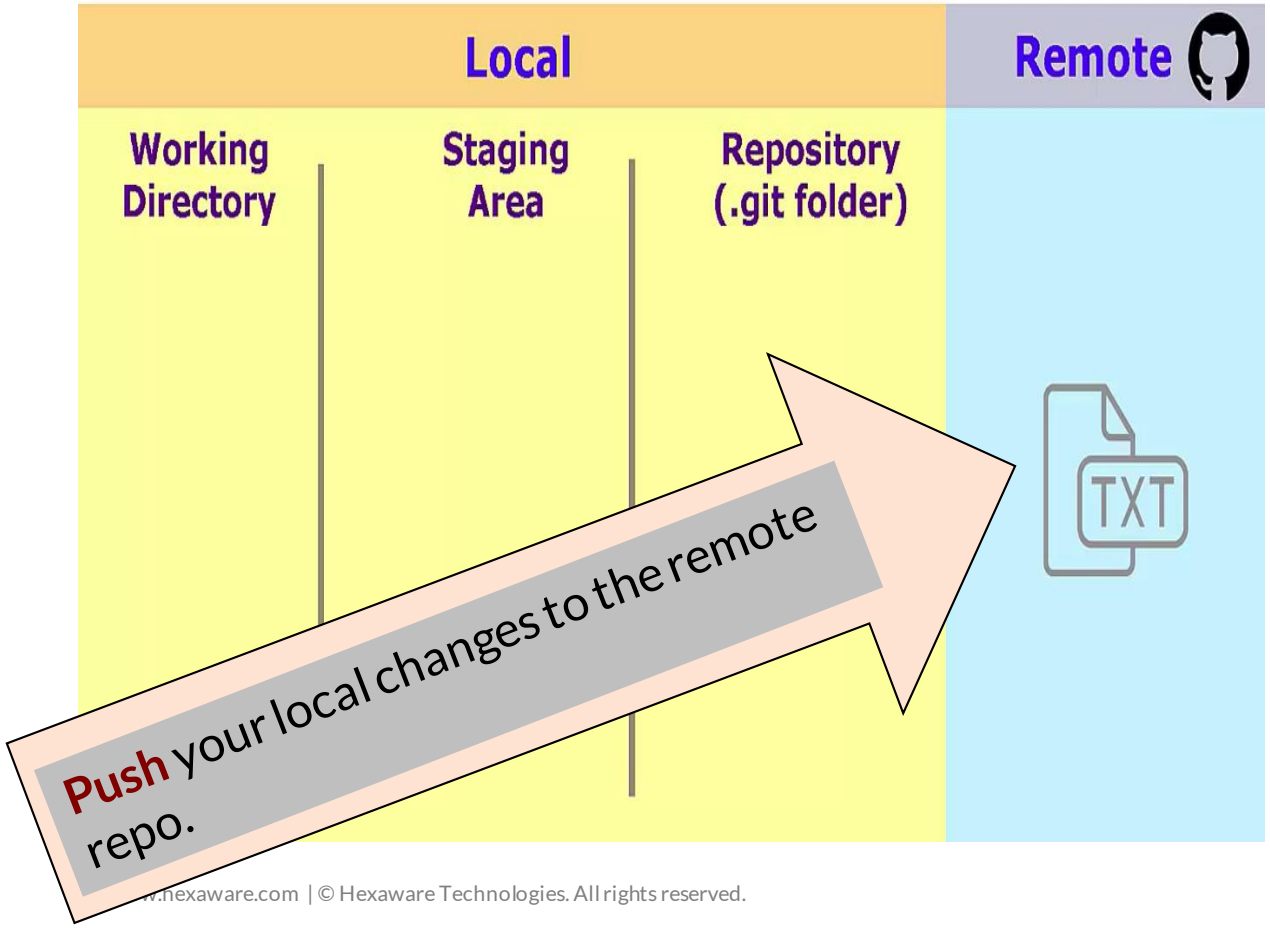
Basic Git Workflow Life Cycle (Cont..)




```
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.

Undo changes on a file

- To undo changes on a file before you have committed it:
  - **git checkout -- filename (undoes your 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 ar

Undo changes on a file (cont.)

- git reset HEAD – filename
  (unstages the file)

```
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)
$
```

Viewing History and diffs

```
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 and diffs (Cont..)

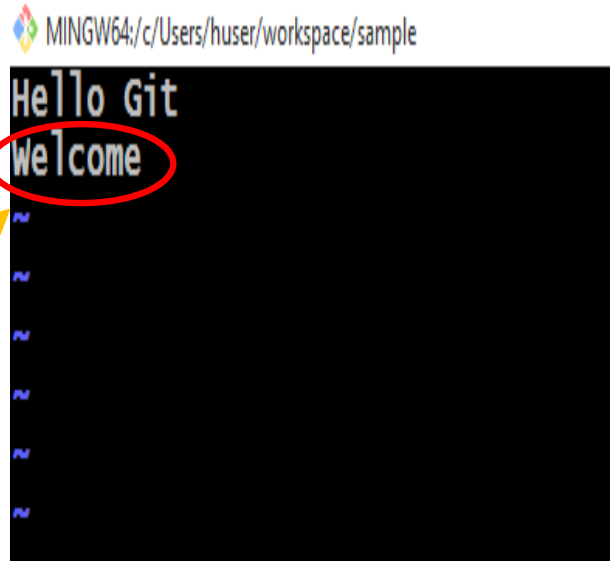
```
37056@HVLAP-07 MINGW64 /c/Users/huser/workspace/sample (master)
$ vi myfile.txt

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 4c40886c0vee7e904babdb376cf7d867139ce3ed
Author: jamunarani <jamunaranik@hexaware.com>
Date:   Tue Mar 20 12:23:13 2018 +0530

    myfile.txt added
```

Viewing History and diffs cont..

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

```
370056@HVLAP-07 MINGW64 /c/Users/huser/workspace (master)
$ echo "Hello git" >> Test.txt

370056@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

370056@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.

370056@HVLAP-07 MINGW64 /c/Users/huser/workspace (master)
$ git diff

370056@HVLAP-07 MINGW64 /c/Users/huser/workspace (master)
$ vi Test.txt

370056@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
```



Viewing History and 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
```

Branching

Git uses branching heavily to switch between multiple tasks.

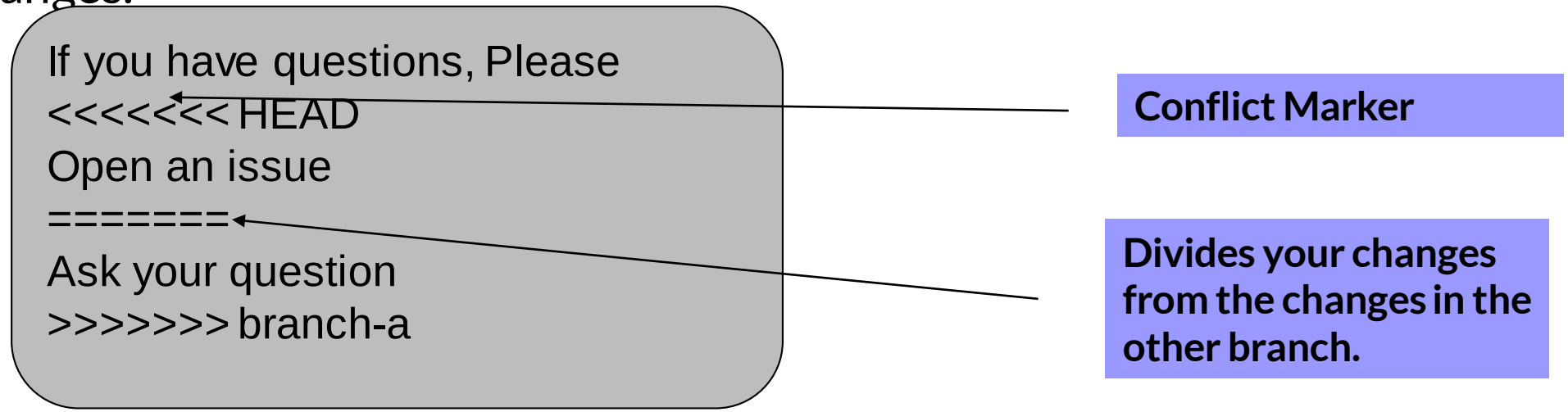
- To create a new local branch:
  - git branch *name*
- To list all local branches: (* = current branch)
  - git branch
- To switch to a given local branch:
  - git checkout branchname
- Creates and checkouts to a branch called <name>
  - git checkout -b <name>

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.

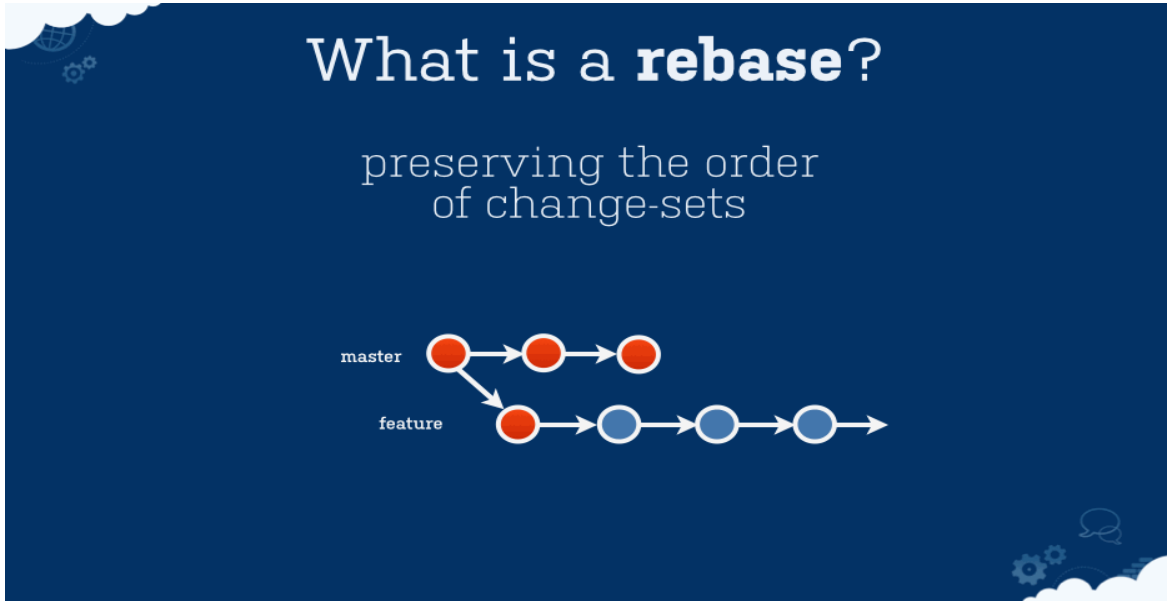


Rebase

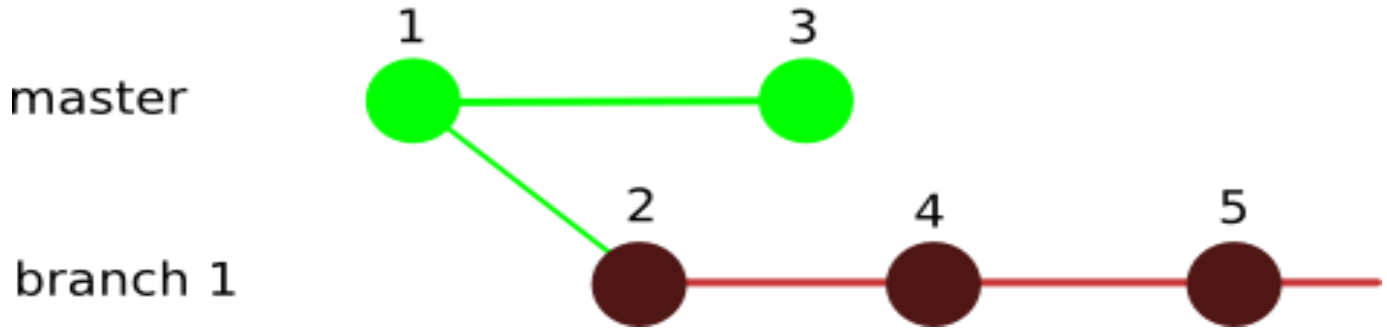
- Changes the "starting point" of a branch
- Can be used in feature branches when the "development branch" changes drastically and it affects the feature branches
- Usage:

  $ git rebase <branch>

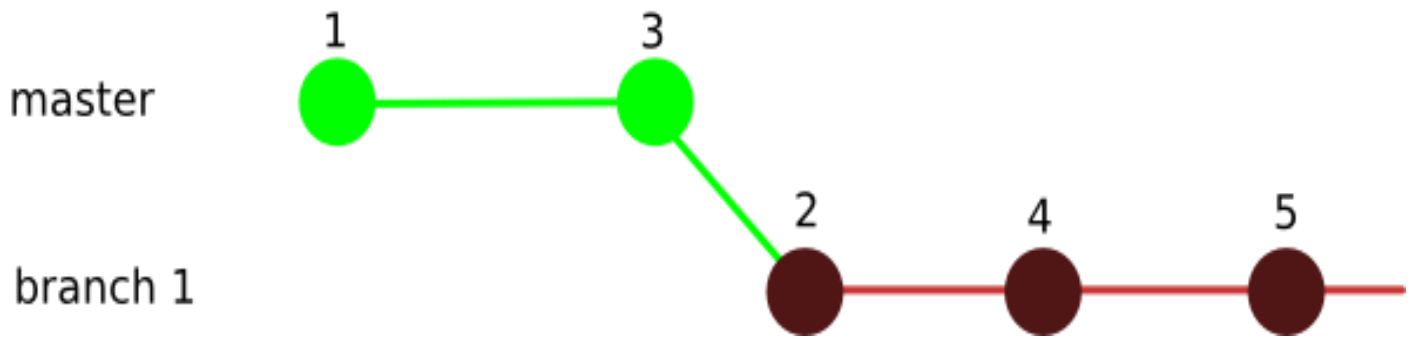
  sets the starting point of the current branch to <branch>



before rebase



after rebase

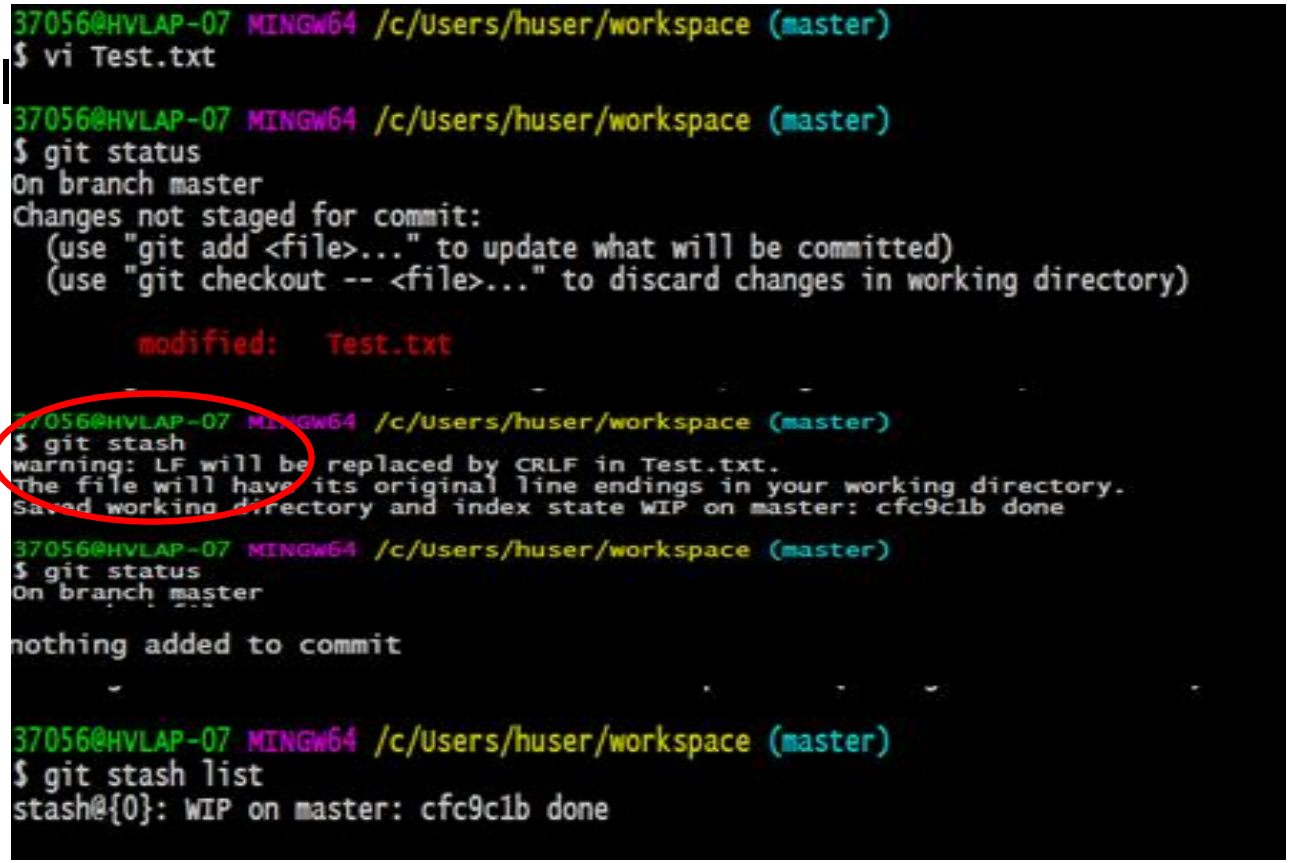


Interaction w/ remote repo

- **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 pul
    - **$ git stash**
  - Listing all stashed changes
    - **$ git stash list**
  - Recovering changes from stash
    - **$ git stash pop/apply**
  - Cleaning up your stash
    - **$ git stash drop**



Best practices for code collaboration

When to commit?

- Source of major arguments (big changes vs small change)
- **Never** put broken code on the master branch (test first!)
- Try not to break things (don't do two weeks worth of work in one commit)
- **Always** use a clear, concise commit message
  - Put more details in lines below, but always make the first line short
  - Describe the *why*; the what is clear in the change log
- When making giant changes, consider branches (we'll talk about these in a few slides)
- Oh, and make sure your name and email are right

- **The Task:**
  1. Create the branch called greeting and check it out.
  2. Edit the greeting.txt to contain your favorite greeting.
  3. Add greeting.txt files to the staging area
  4. Commit
  5. Switch back to the master branch
  6. Create a file README.md with information about this repository
  7. Add the README.md file to staging area and make the commit
  8. Diff the branches
  9. Merge the greeting branch into master

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