

Instructor Notes:

Add instructor notes here.

DevOps

Lesson 03- GIT

Instructor Notes:

Add instructor notes here.

Lesson Objectives

- Introduction to GIT
- Version Control
- Repositories and Branches
- Working Locally with GIT
- Working Remotely with GIT



Instructor Notes:

Add instructor notes here.

3.1: Introduction to Git

Introduction

- Initially developed by Linus Torvalds, Git is a distributed version control system
- Git is a distributed revision control and source code management (SCM) system with an emphasis on speed, data integrity and support for distributed, non-linear workflows
- As with most other distributed revision control systems, and unlike most client-server systems, every Git working directory is a full-fledged repository with complete history and full version-tracking capabilities, independent of network access or a central server. Like the Linux kernel, Git is free software distributed under the terms of the GNU General Public License version 2



Copyright © Capgemini 2016. All Rights Reserved 3

Add the notes here.


Instructor Notes:


Add instructor notes here.

3.1: Introduction to Git

Introduction

- Design Philosophy
 - Free & Open Source
 - Blazingly Fast
 - Distributed
 - Data Assurance
 - Strong support for non-linear development
 - Compatibility with existing systems/protocols
 - Toolkit-based design





Copyright © Capgemini 2016. All Rights Reserved 4

Blazingly Fast

Torvalds has described Git as being very fast and scalable and performance tests done by [Mozilla](#) showed it was an [order of magnitude](#) faster than some version-control systems, and fetching version history from a locally stored repository can be one hundred times faster than fetching it from the remote server

Distributed development

Like [Darcs](#), [BitKeeper](#), [Mercurial](#), [SVK](#), [Bazaar](#) and [Monotone](#), Git gives each developer a local copy of the entire development history, and changes are copied from one such repository to another. These changes are imported as additional development branches, and can be merged in the same way as a locally developed branch.

Data Assurance

Aborting operations or backing out changes will leave useless dangling objects in the database. These are generally a small fraction of the continuously growing history of wanted objects. Git will automatically perform [garbage collection](#) when enough loose objects have been created in the repository.

Garbage collection can be called explicitly using `git gc --prun`.

Strong support for non-linear development

Git supports rapid branching and merging, and includes specific tools for visualizing and navigating a non-linear development history. A core assumption in Git is that a change will be merged more often than it is written, as it is passed around various reviewers.

Branches in git are very lightweight: A branch in git is only a reference to a single commit. With its parental commits, the full branch structure can be constructed.

Instructor Notes:

Add instructor notes here.

3.2: Version control

Version Control System

- Version control systems are software that manage changes to files e.g. documents, images, code etc.
- Benefits of Version Control
 - Saves from creating multiple backup files
 - Allow multiple people to work on same file
 - Track changes & see who had made changes
 - Easy to switch back to older version when needed
 - Increases productivity
- Two types of Version Control
 - Client sever Version Control- SVN, CVS, PerForce, IBM rational
 - Distributed Version Control -GIT

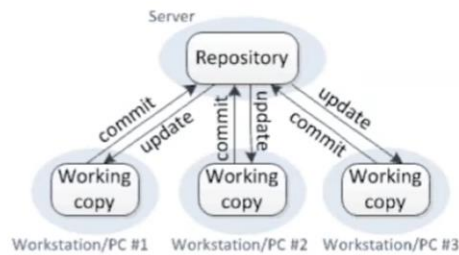
Instructor Notes:

Add instructor notes here.

3.2: Version control

Problem with Client Server Version Control

- Client Server version control system works on a centralized model which has a single repository to which user check-in & check-out
- Some of the major benefits working with version control system are listed below:
 - Version control is not available on local system
 - If the central server get corrupted the entire history is lost



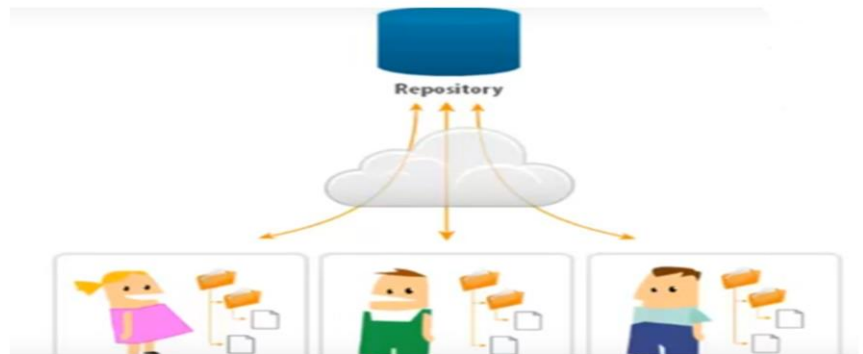
Instructor Notes:

Add instructor notes here.

3.2: Version control

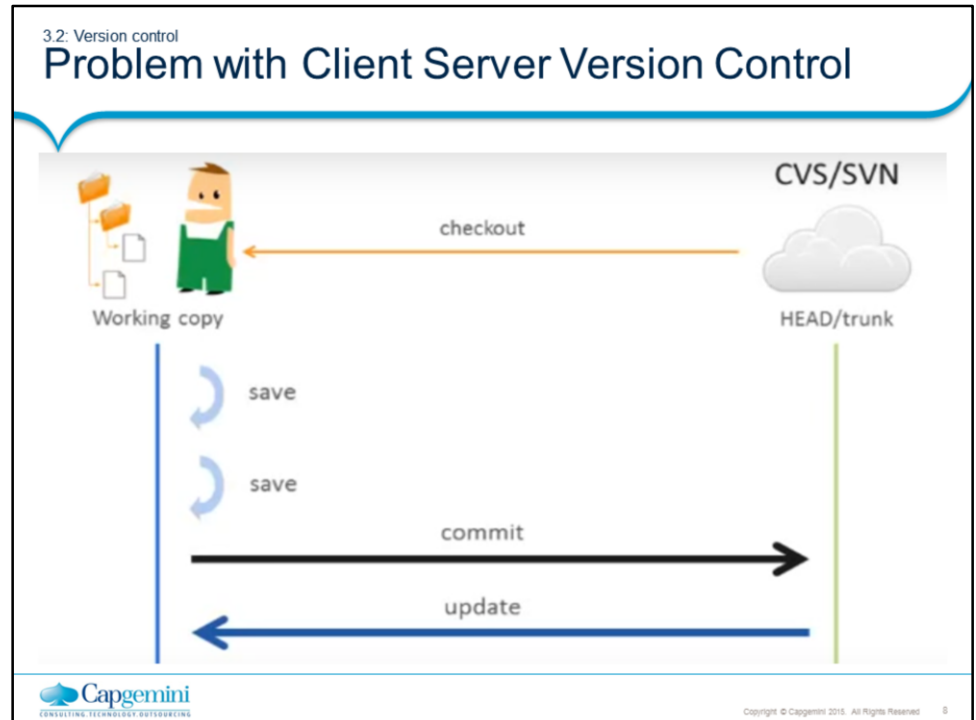
Problem with Client Server Version Control

- History in one Repository
- Client only gets a single revision per checkout
- All commits go into one repository



Instructor Notes:

Add instructor notes here.



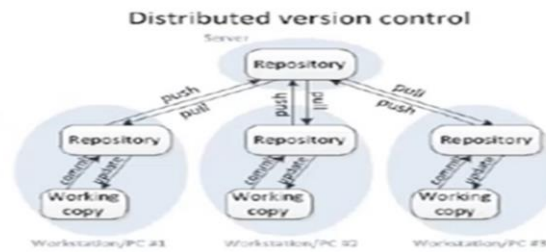
Instructor Notes:

Add instructor notes here.

3.2: Version control

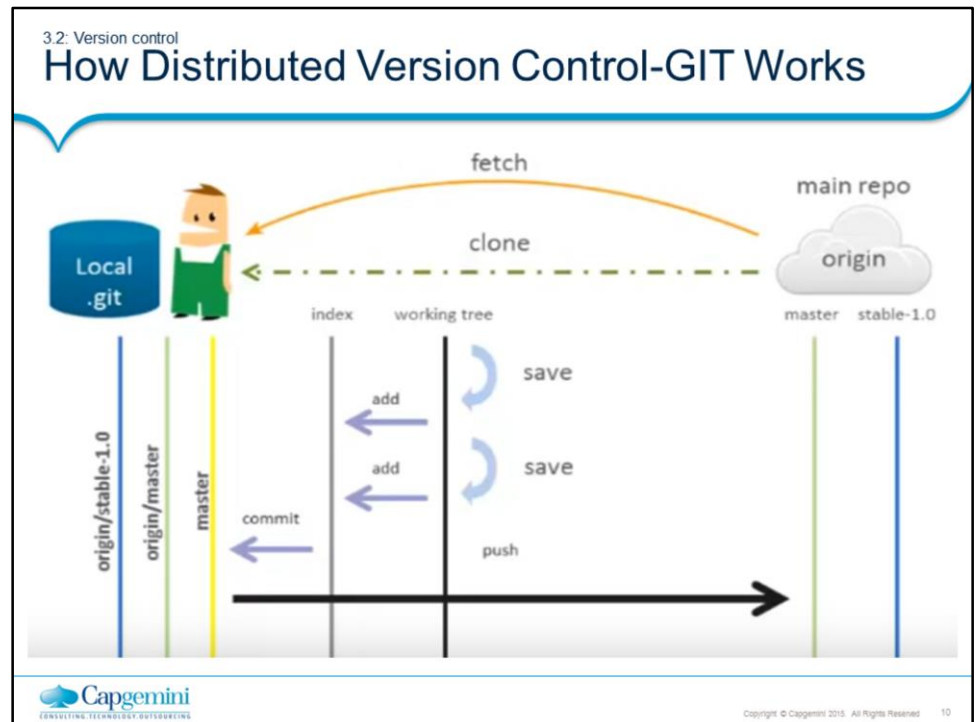
Advantage of Distributed Version Control-GIT

- Distributed version control system don't rely on central server. It allows one of the repository on their own drive with entire history of the project
- Benefits of Distributed version control :
 - DVCS is also available on local machine.
 - No single point of failure as each user has the repository with entire history
 - Performs all action locally , even when not connected to internet



Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

3.2: Version control

Version Control Parallel Development -GIT

- Version control system helps in parallel development and preventing one user from overwriting the work of another
- Two ways to solve parallel development problem:
 - Copy-modify-merge –GIT uses this
 - Lock-modify-unlock(practically not possible)

Instructor Notes:

Add instructor notes here.

3.2: Version control

Version Control Parallel Development -GIT

- User A commits a new version to their local repository.
- User B commits a new version to their local repository.

Repository

User A

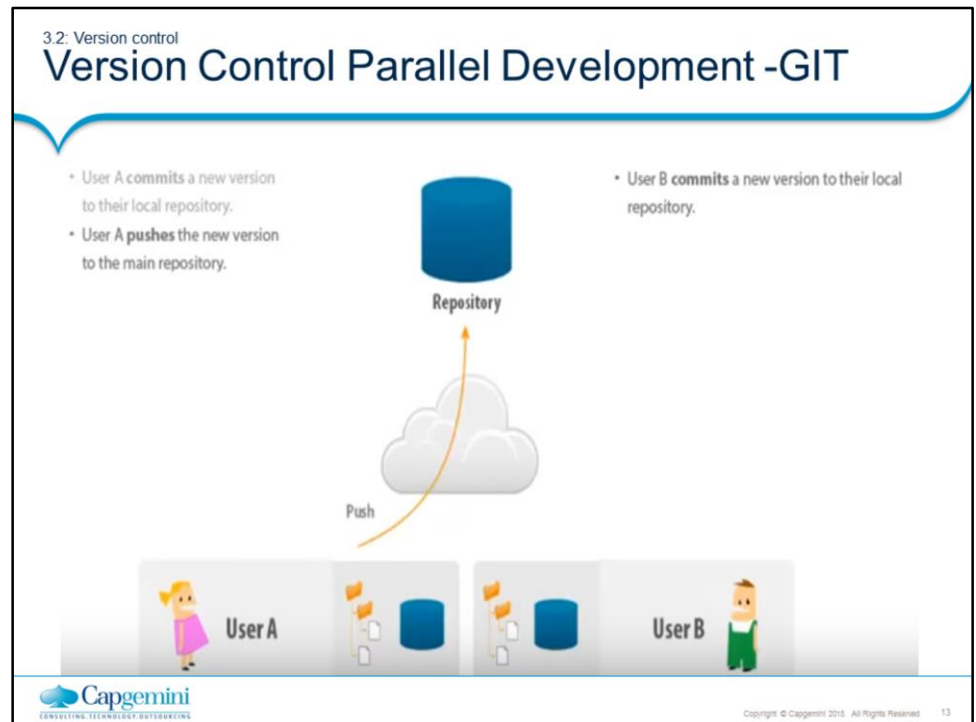
User B

Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2016. All Rights Reserved 12

Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

3.2: Version control

Version Control Parallel Development -GIT

- User A commits a new version to their local repository.
- User A pushes the new version to the main repository.

The diagram shows a central 'Repository' (blue cylinder) at the top. Below it is a cloud icon with a red 'X' and an orange arrow pointing from the cloud to the Repository, labeled 'Push'. At the bottom, there are two user boxes: 'User A' on the left and 'User B' on the right. Each user box contains a local repository icon (blue cylinder) and a file icon. User A's local repository is connected to the main Repository by a green arrow. User B's local repository is connected to the main Repository by a green arrow. The orange arrow from the cloud to the main Repository indicates a failed push attempt by User B.

- User B commits a new version to their local repository.
- User B tries to **push** the new version to the main repository, but it fails indicating he needs to update the local version first.

Repository

Push

User A

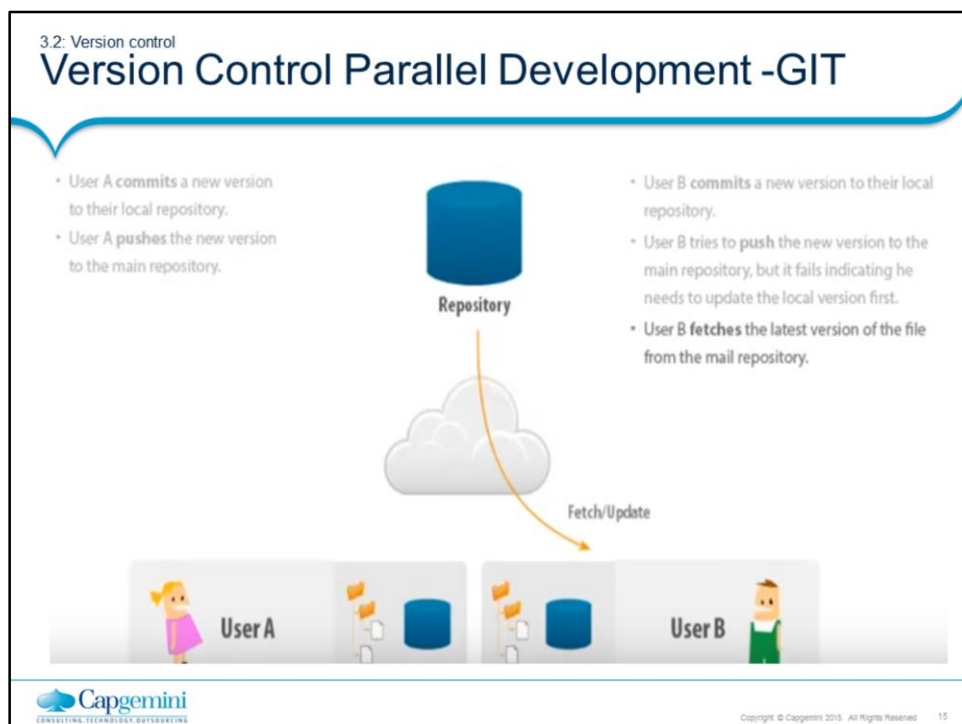
User B

Capgemini
CONSULTING TECHNOLOGY ENTREPRENEURS

Copyright © Capgemini 2016. All Rights Reserved 14

Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

3.2: Version control

Parallel Development, Copy Modify Merge

The diagram illustrates a workflow for parallel development using version control. It shows two users, User A and User B, interacting with a central Repository (represented by a blue cylinder) and their local repositories (represented by blue cylinders with folders). The workflow is as follows:

- User A commits a new version to their local repository.
- User A pushes the new version to the main repository.
- User B commits a new version to their local repository.
- User B tries to push the new version to the main repository, but it fails indicating he needs to update the local version first.
- User B fetches the latest version of the file from the main repository.
- User B merges changes into his local version of the file.
- User B commits the combined into his local repository.
- User B pushes the merged version of the file to the main repository.

The diagram shows User A pushing to the Repository, and User B fetching from the Repository, merging, and then pushing back to the Repository.

Capgemini
CONSULTING TECHNOLOGY ENTERPRISES

Copyright © Capgemini 2016. All Rights Reserved 10

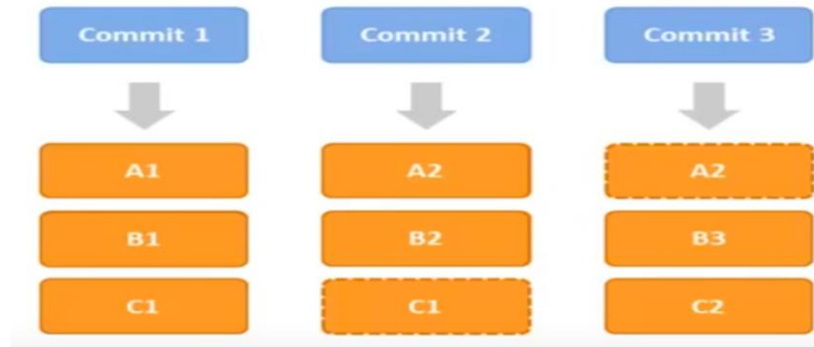
Instructor Notes:

Add instructor notes here.

3.2: Version control

Git – Snapshot Storage

- Snapshot Storage –stores the complete files changed by a commit along with references to files that were not changed by that commit.



Instructor Notes:

Add instructor notes here.

3.3: Repositories and Branches

GIT-Repositories

- **Repositories:**

- It is a collection of refs together with an object database containing all objects which are reachable from the refs, possibly accompanied by meta data from one or more porcelain. A repository can share an object database with other repositories via alternate mechanism.

- **What to store in repositories?**

- Anything, however any sort of editable files are preferred.

Instructor Notes:

Add instructor notes here.

3.3: Repositories and Branches

Git – Snapshot Storage

- Creating repositories :

- at default location
 - git init
- at particular location
 - git init c:/testGIT
- Bare repository
 - git init --bare

- How to get GIT repository:

- `$ git clone git://git.kernel.org/pub/scm/git/git.git`

It does approx. 225 MB download

Instructor Notes:

Add instructor notes here.

3.3: Repositories and Branches

Repositories and Branches

■ Branches:

- A "branch" is an active line of development.
- The most recent commit on a branch is referred to as the tip of that branch.
- The tip of the branch is referenced by a branch head, which moves forward as additional development is done on the branch.
- A single git repository can track an arbitrary number of branches, but working tree is associated with just one of them (the "current" or "checked out" branch), and HEAD points to that branch



Copyright © Capgemini 2016. All Rights Reserved 20

A bare repository is normally an appropriately named [directory](#) with a .git suffix that does not have a locally checked-out copy of any of the files under revision control. That is, all of the git administrative and control files that would normally be present in the hidden .git sub-directory are directly present in the repository. Git directory instead, and no other files are present and checked out. Usually publishers of public repositories make bare repositories available. master The default development [branch](#). Whenever you create a git [repository](#), a branch named "master" is created, and becomes the active branch. In most cases, this contains the local development, though that is purely by convention and is not required.

Instructor Notes:

Add instructor notes here.

3.3: Repositories and Branches

Repositories and Branches

- Getting different versions of project:

- Git is best thought of as a tool for storing the history of a collection of files. It stores the history as a compressed collection of interrelated snapshots of the project's contents. In git each such version is called a commit.
- Those snapshots aren't necessarily all arranged in a single line from oldest to newest; instead, work may simultaneously proceed along parallel lines of development, called branches, which may merge and diverge.
- A single git repository can track development on multiple branches. It does this by keeping a list of heads which reference the latest commit on each branch; the `git-branch(1)` command shows you the list of branch heads:
 - `$ git branch * master`
- A freshly cloned repository contains a single branch head, by default named "master", with the working directory initialized to the state of the project referred to by that branch head.
- Most projects also use tags. Tags, like heads, are references into the project's history, and can be listed using the `git-tag(1)` command:
 - `$ git tag -l`



Copyright © Capgemini 2016. All Rights Reserved 21

`git branch`

List all of the branches in our repository.

`git branch <branch>`

Create a new branch called <branch>. This does *not* check out the new branch.

`git branch -d <branch>`

Force delete the specified branch

`git branch -m <branch>`

Examples:

It's important to understand that branches are just *pointers* to commits. When you create a branch, all Git needs to do is create a new pointer—it doesn't change the repository in any other way.

create a branch using the following command:

`git branch branch-experiment`

The repository history remains unchanged. All you get is a new pointer to the current commit

Once we are finished working on a branch and have merged it into the main code base, we are free to delete the branch without losing any history

`git branch -d branch-experiment`

However, if the branch hasn't been merged, the above command will output an error message

`git checkout <existing-branch>`

Check out the specified branch, which should have already been created with `git branch`. This makes <existing-branch> the current branch, and updates the working directory to match.

`git checkout -b <new-branch>`

Create and check out <new-branch>. The `-b` option is a convenience flag that tells Git to run `git branch <new-branch>` before running `git checkout <new-branch>`. `git checkout -b <new-branch> <existing-branch>`

Same as the above invocation, but base the new branch off of <existing-branch> instead of the current branch.

Instructor Notes:

Add instructor notes here.

3.3: Repositories and Branches

Understanding History-Repositories

- **Commits:**

- Every change in the history of a project is represented by a commit. The `git-show(1)` command shows the most recent commit on the current branch:
- `$ git show`
- Every commit (except the very first commit in a project) also has a parent commit which shows what happened before this commit. Following the chain of parents will eventually take you back to the beginning of the project.
- However, the commits do not form a simple list; git allows lines of development to diverge and then reconverge, and the point where two lines of development reconverge is called a "merge". The commit representing a merge can therefore have more than one parent, with each parent representing the most recent commit on one of the lines of development leading to that point.
- The best way to see how this works is using the `gitk(1)` command; running `gitk` now on a git repository and looking for merge commits will help understand how the git organizes history.
- In the following, commit X is "reachable" from commit Y if commit X is an ancestor of commit Y. Equivalently, Y is a descendant of X, or that there is a chain of parents leading from commit Y to commit X.

Instructor Notes:

Add instructor notes here.

3.3: Repositories and Branches

Understanding History-Trees

- The working tree is the current view into the repository. It has all the files from your project: the source code, build files, unit tests, and so on.
- Some VCSs refer to this as your working copy. People coming to Git for the first time from another VCS often have trouble separating the working tree from the repository. In a VCS such as Subversion, your repository exists “over there” on another server.
- In Git, “over there” means in the `.git/` directory inside your project’s directory on your local computer. This means you can look at the history of the repository and see what has changed without having to communicate with a repository on another server.

Instructor Notes:

Add instructor notes here.

3.4: Working Locally with GIT

Locally Working with GIT

- Download GIT From <https://git-scm.com/downloads>
- First Repository :
 - Create Empty directory on system myrepoone
 - Create a repository
git init

```
r vikash@PUNHDCLT58 MINGW32 ~/Desktop
$ cd myrepoone
bash: cd: myrepoone: No such file or directory

r vikash@PUNHDCLT58 MINGW32 ~/Desktop
$ cd C:\myrepoone

r vikash@PUNHDCLT58 MINGW32 /c/myrepoone
$ git init
Initialized empty Git repository in C:/myrepoone/.git/

r vikash@PUNHDCLT58 MINGW32 /c/myrepoone (master)
$ ls -a
./ ../ .git/
```

Checking repository created or not

Instructor Notes:

Add instructor notes here.

3.4: Working Locally with GIT

Locally Working with GIT

- Setting Name & email Id

- Setting name

```
git config --global user.name "Rahul Vikash"
```

- Setting email

```
$ git config --global user.mail rahul.vikash@capgemini.com
```

- Checking configuration

```
$ git config --list
```

```
vikash@PUNHDC1T58 MINGW32 /c/myrepoone (master)
$ git config --global user.name "Rahul Vikash"
vikash@PUNHDC1T58 MINGW32 /c/myrepoone (master)
$ ^C
vikash@PUNHDC1T58 MINGW32 /c/myrepoone (master)
$ git config --global user.mail rahul.vikash@capgemini.com
vikash@PUNHDC1T58 MINGW32 /c/myrepoone (master)
$ ^C
vikash@PUNHDC1T58 MINGW32 /c/myrepoone (master)
$ git config --list
core.symlinks=false
core.autocrlf=true
core.fsmonitor=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
pack.packsize=1024
http.postBuffer=524288
http.sslCAinfo=C:/Program Files/Git/mingw32/ssl/certs/ca-bundle.crt
rebase.autosquash=true
credential.helper=manager
user.name=Rahul Vikash
user.mail=rahul.vikash@capgemini.com
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.ignorecase=true
core.sshCommand=ssh
```



Instructor Notes:

Add instructor notes here.

3.4: Working Locally with GIT

Locally Working with GIT

- Adding File in repository

MyFirst.txt

- Check

\$ ls

- Know the status

\$ git status

- Add the file in staging

\$ git add MyFirst.txt

- Commit in git repository with lock message

\$ git commit -m "Added MyFirst File"

- See log

\$ git log



Copyright © Capgemini 2016. All Rights Reserved 26

Instructor Notes:

Add instructor notes here.

3.4: Working Locally with GIT

Locally Working with GIT

```
r vikash@PUNHDCLT$ MINGW32 /c/myrepoone (master)
$ git commit -m "Added MyFirst File"
[master (root-commit) d06ddf1] Added MyFirst File
Committer: Rahul Vikash <rahul.vikash@capgemini.com>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:
```

```
git config --global user.name "Your Name"
git config --global user.email you@example.com
```

After doing this, you may fix the identity used for this commit with:

```
git commit --amend --reset-author
```

```
1 file changed, 1 insertion(+)
create mode 100644 MyFirst.txt
```

Instructor Notes:

Add instructor notes here.

3.4: Working Locally with GIT

Locally Working with GIT-

Adding new data in MyFirst.txt

Now checking status \$git status

```
rvikash@PUNHDCLT58 MINGW32 /c/myrepoone (master)
$ 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:   MyFirst.txt

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

Commit \$ git commit -am "Added New MyFirst File"

Again check log --\$git log



Copyright © Capgemini 2016. All Rights Reserved 28

Instructor Notes:

Add instructor notes here.

3.4: Working Locally with GIT

Locally Working with GIT-

- To check which line has changed
\$ git commit - -amend
- Add one more file -MyJava.txt
- Unstage the One File
\$ git reset HEAD MyFirst.txt
- Unchanged the work done --\$ git checkout -- MyJava.txt

```
rvikash@PUNHDCLT58 MINGW32 /c/myrepoone (master)
$ git checkout -- MyJava.txt

rvikash@PUNHDCLT58 MINGW32 /c/myrepoone (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   MyJava.txt
```

Instructor Notes:

Add instructor notes here.

3.4: Working Locally with GIT

Locally Working with GIT

- Multiple Reset for n number of file ,Goes to last commit
\$ git reset -- hard

Instructor Notes:

Add instructor notes here.

3.5: Working Remotely with GIT

Remotely Working with GIT

- Cloning the data ----git clone <repo> <directory>

```
$ git clone rahul.vikash@capgemini.com:myrepoone
```

- git-remote - Manage set of tracked repositories, Manage the set of repositories ("remotes") whose branches you track.
- git-push - Update remote refs along with associated objects, Updates remote refs using local refs, while sending objects necessary to complete the given refs.

Instructor Notes:

Add instructor notes here.

3.5: Working Remotely with GIT

Remotely Working with GIT

- git-fetch - Download objects and refs from another repository
- git-pull - Fetch from and integrate with another repository or a local branch
- git-merge - Join two or more development histories together

Instructor Notes:

Add instructor notes here.

Demo

- Demo on GIT –Locally using git init, add, status, log
- Demo on GIT- Remotelly using git remote, pull, push, fetch, clone



Copyright © Capgemini 2016. All Rights Reserved 33

Add the notes here.

Instructor Notes:

Add instructor notes here.

Lab

■ Lab 01



Add the notes here.

Instructor Notes:

Add instructor notes here.

Summary

- Git is a distributed revision control and source code management (SCM) system with an emphasis on speed, data integrity and support for distributed, non-linear workflows.
- Git working with local repository
- Git working with remote repository



Add the notes here.

Instructor Notes:

Q1 git diff master branch
name

Q2 pull

Q3

Review Question

- To see the difference between which two branches the following command can be used?
 - git diff master branch_name
 - git --diff master branch_name
 - git merge master branch_name
 - git --stat master branch_name
- The git _____ command performs a git fetch and git merge.
 - Push
 - Pull
 - Clone
 - branch



Copyright © Capgemini 2016. All Rights Reserved 36

Add the notes here.

Instructor Notes:

Q3 \$ git add file1 file2
file3->\$ git diff --cached-
>\$ git status->\$ git
commit

Review Question

- Three files file1, file2 and file3 are modified. Identify the series of commands to view the modified files, then add their updated contents to the index and commit the changes.
 - '\$ git status ->\$ git commit
 - \$ git diff --cached->\$ git status->\$ git commit -a
 - \$ git add ->\$ git diff --cached->\$ git status->\$ git commit
 - \$ git add file1 file2 file3->\$ git diff --cached->\$ git status->\$ git commit



Add the notes here.