**GITHUB**

Rm -rf folder-name 🡪 remove folder forcefully

Ls 🡪 list of folders

Git clone 🡪 cloning the items

Vi 🡪 it’s an editor

Escape +: q 🡪 for close editor

Escape +: wq 🡪 for save and close editor

Escape +: wq! 🡪 for forcefully save and quit

Touch 🡪 used to create folder and files

Git add “file-name” 🡪 adding one (or) specific file

Git status 🡪 used to view status of the files

Git config –-global user.name “user-name”

Git config –-global user.email” email-id”

Git config –-global user

Git rm –-cached hello.java 🡪 remove file from staging area

**Git stash 🡪** reverting locally deleted file

**Git prune 🡪**

**Git revert deleted log-id 🡪** reverting centrally deleted file

**git revert reverted log-id 🡪** deleting reverted file

**git commit -m “information about file”** 🡪adding file to local repository

[master (root-commit) 45a3529] added hello.java file

1 file changed, 1 insertion(+)

create mode 100644 hello.java

**git log** 🡪 information about commit id, author and committing message with date and time

commit 45a3529386e513f9e4284c3243b8ce7dfc979ca9 (HEAD -> master)

Author: abdul <shaikabdulubed@gmail.com>

Date: Sat Feb 29 19:13:14 2020 +0530

added hello.java file

**git push origin master** 🡪 pushing our local repository files to master repository

Enumerating objects: 3, done.

Counting objects: 100% (3/3), done.

Writing objects: 100% (3/3), 224 bytes | 224.00 KiB/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To https://github.com/noorabdulubed/Test-Repo1.git

\* [new branch] master -> master

**Git config –-list 🡪** shows all configuration files ex-user.name and user.email

**git checkout commit-id 🡪**  switching between commit id’s

**User Access Control**

1. Create organizations (project) ex: “test-organization”
2. Add/create repositories into organization ex: “test-organization”
3. Create users and add users into organization
4. Control permissions of users
   1. Dev1 🡪 write (commit)
   2. Dev2 🡪 read (clone/pull)
   3. User1 🡪 owner of organization
   4. User2 🡪 admin of organization
   5. Abdul 🡪 admin/super user

**GIT**

* Create one folder in any one operating system (**Ubuntu)** and set that folder as a server by using **git init --bare** command 🡪 created empty repository folder
* Create one folder in any one operating system (**windows)** and clone repository folder by using **git clone username@ip-addr:/complete path of the repository folder**
* **Ex: git clone**

**[abdulubed@192.168.0.9:/home/abdulubed/Desktop/Git-Server](mailto:abdulubed@192.168.0.9:/home/abdulubed/Desktop/Git-Server)**

* By connecting VMware virtual machine to putty we get error like **connection refused** we need to install **open-sshserver**
  + **Command: sudo apt-get install openssh-server**
* To check repository url and other information then goto repository filder inside repository folder goto **.git** folder then type below command
  + **Command: vi config**

**Error:**

* Remote unpack failed : unable create temporary object failed to push

**Solution:**

It sounds like you have file in the git repo owned by root. Since you're ssh'ing in as 'username' to do the push, the files must be writable by username. The easiest thing is probably to create the repo as the user, and use the same user to do your pushes. Another option is to create a group, make everything writable by the group, and make your user a member of that group.

(or)

We need to give correct path

**What is Conflict**

When ever we update local repository update will be a failure because

* If the file is modified by multiple users, I mean that to commonly same lines also at the time update my local repository update will be failure
* Conflict will not happen for repository it will happen for file that to lines in that file

**Configure merging tool**

* git config --global core.editor "vim"
* git config --global merge.tool "vimdiff"

**opeing merge tool**

* git mergetool

**Branches**

* master branch 🡪 integrated application source code 🡪 final source code🡪 only one master branch
* feature branch 🡪 for developers purpose 🡪multiple feature branch
* **git branch -a 🡪** list of available local and remote branches
  + **red color** represents remote branches 🡪 **git branch -r**
  + **green color** represents local branches 🡪 **git branch -l**
* **create branch**
  + **git branch branch name**
    - **Ex: git branch qa** 🡪 **c**reate **qa** branch 🡪 **block color**
* **Git checkout branchName 🡪checkout** is used to switch between branches
  + **Ex: git checkout qa 🡪** switched branch qa

**Merging**

* Merging done by code reviewers
* Code reviewers reviewed source code and merge source code into master branch

**Creating branches without linking from existing branch** 🡪 **orphan branch**

* **Git checkout –-orphan branch-name**
  + **Ex: git checkout –-orphan test-branch 🡪** created test-branch with **existing data** but **without copying existing logs**
  + if orphan branch (or) any branch doesn’t contain any commit id’s (or) logs it doesn’t push that branch to origin (or) remote
* why do we create orphan branch?
  + on this orphan branch developers don’t maintain project code instead they are committing the non-project related files (developers do some experiments then they are used this orphan branches)
* how to merge branches
  + **git merge branch name** 
    - Ex: we are in master branch now we need to merge testing branch with master branch then command is
      * **Git merge testing**
* if we merge two branches for example: master branch and testing branch
* we are now in master branch and perform merge operation
  + **git merge testing**
* then in master branch latest commit id is from testing branch
* if we merge first time then merge commit id is not generated second time onwards merge commit id is generated along with message **merge branch ‘merged branch name’**

**Rebase:**

* Git Rebase is same as Git Merge only difference is Git Merge is not followed sequential order to show merge commit id’s and Git Rebase followed sequential order to show merge commit id’s
* Git Merge is refused to merge orphan branches
* Git Rebase is allow to merge orphan branches but deletes all old files

**Linux**

Linux user-name is 🡪 **root** (or) **su** (super user)

Types of servers (or) instances

1. Physical server
2. Virtual server
3. Cloud server
4. Docker containers

**Wind. Hypervisor** tool 🡪 used to create virtual machines 🡪 Ex: VMware-work station, oracle-virtual box, Hyper-V

**Linux commands:**

* cd 🡪 change directory
* cd / 🡪 slash is system root directory (or) OS root directory
* cd ~ 🡪 tiled represents user home directory
* pwd 🡪 present working directory
* ls 🡪 listing files and directories
* bin folder contains binaries 🡪 all commands
* cd .. 🡪 one back to present working directory
* su 🡪 switch user
* non super users are in **/home** directory
* super user directory under **/root** directory
* su non-super user-name 🡪 redirected to non-super user
  + ex: su ec2-user
* mkdir 🡪 create a directory
* touch 🡪 create files
* vi 🡪 to modify files like a notepad
* cat 🡪 to view files
* cp 🡪 used to copy files
  + cp source-file /destination-path
  + ex 🡪 [root@ip]# cp file1.java /test(destination)
* mv 🡪 used to movie files and folders
  + mv source-file /destination-path
  + ex 🡪 [root@ip]# mv file1.java /qa(destination)
* copying folders 🡪 folders are copied by recursively
  + cp -r source-file /destination-path
  + ex 🡪 cp -r development /test
* delete files
  + rm file-name
  + ex 🡪 rm file1.java
* delete files without specifying yes option
  + rm -f file-name (or) folder-name
  + rm -r file1.java
* delete directories
  + rm -r directory-name
  + ex 🡪 rm -r development
* delete directories with specifying yes option
  + rm -rf directory-name
  + ex 🡪 rm -rf development
* rename
  + mv old-file-name new-file-name
  + ex 🡪 mv file1.java file5.java 🡪 file1.java becomes file5.java
* service(demon)
  + an application which is running in the background
  + service is also called demon
  + when we reboot our system, I want my service to be running by default enable-it
  + when we reboot our system, I want my service to be stopping by default disable-it
  + RHEL-6 firewalls
    - iptables
    - ip6tables
  + RHEL-7 firewalls
    - Firewalld
  + To check status of the service
    - Service service-name status
* yum install iptables-services
* yum install firewalld
* ~]# systemctl start iptables
* ~]# systemctl start iptables
* ~]# systemctl stop ip6tables
* ~]# systemctl enable iptables
* ~]# systemctl disable iptables
* ~]# systemctl enable ip6tables
* To check ip-address
  + Ifconfig
* To check network related services
  + Root to 🡪 /etc/sysconfig/network-scripts
* How to restart service
  + Service service-name restart
    - Ex: service NetworkManger restart

**File Permissions**

File1.txt 🡪 read-r(4), write-w(2), exe-x(1)

File1.txt 🡪 rwx..4+2+1=7

r-x 🡪 4+1=5

--x 🡪 +1=1

r-- 🡪 =4

* Ls -l 🡪 show permissions (ls -l 🡪 long list)
  + Ex: rw-(owner) --(groups) r--(others)
* Change permissions to files
  + Chmod 🡪 changing the mode of the file
    - Ex🡪 chmod 541 file1.txt [r-x(5) r—(4) –x(1)]
* Default file permission 🡪 644
* Default directory permission 🡪 755
* Change folder permissions along with folder contains files change permissions using recursively
  + Ex 🡪 Chmod -R 777

**Create users:**

Users are under **etc/passwd** directory

* Creating user 🡪 **useradd abdul**
* Assign password to the user 🡪 **passwd abdul**

**Creating groups:**

Groups are under **etc/group** directory

* Creating group 🡪 **groupadd group-name**
  + Ex: **groupadd dev-group**
* Adding users to groups 🡪 **usermod -G group-name user-name**
  + Ex: **usermod -G dev-group abdul**

**YUM 🡪 yum** is a repository it consists of list of packages

* By default, yum is available when installed os

**RPM 🡪 RedHatPackageManagement**

* If external packages are not available in the **yum** repository then these packages must be downloaded in **RPM** repository
* The file download as a .**RPM**
* Once external packages downloaded then installed that package by using **rpm**
* **Rpm-qa 🡪** list of packages which are installed under our os
  + Ex: **rpm -ivh chefdk-4.7.73-1.el7.x86\_64.rpm**
    - **-ivh 🡪 install verbose human readable format**
* **Grep 🡪** grep is going to search packages
  + **Ex: rpm -qa | grep chef**

**WGET 🡪** by using **wget** external packages are download

* **Wget url-name download**
  + Ex: **wget chef-dk\_url download**
  + Ex: **rpm -ivh chefdk-4.7.73-1.el7.x86\_64.rpm**
    - **-ivh 🡪 install verbose human readable format**

**Uninstall packages**

* **Rpm -e(erase) package-name** 
  + Ex: rpm -e chefdk-4.7.73-1.el7.x86\_64
* **Yum erase package-name**
  + Ex: yum erase git

**MAVEN**

1. Clear 🡪 pre-clean, clean, post-clean
2. Build(default) 🡪 resource compile test package install
3. Site 🡪 pre-site, site, post-site
4. Mvn compile 🡪 resource -> compile
5. Mvn test 🡪 resource -> compile -> test
6. Mvn package 🡪 resource -> compile -> test -> package
7. Mvn install 🡪 resource -> compile -> test -> package -> install

* Mvn clean
* Mvn clean compile
* Mvn clean package

|  |  |
| --- | --- |
| **ANT** | **MAVEN** |
| 1. Scripts | 1. Commands |
| 1. Build.xml | 1. pom.xml |
| 1. no plugins | 1. plugins |
| 1. no life-cycles | 4.life-cycles |
| 1. non-conventional | 5.conventional |
| 1. no dependencies | 6.dependencies |