## **Git Installation**

* $ sudo dnf install git-all
* $ git –version

## **Git Configuration**

#git config --system /etc/gitconfig [System specific]

#git config --global ~/.gitconfig [user home directory specification]

#git config --local .gitconfig [current repo]

**Your Identity**

* $ git config --global user.name "John Doe"
* $ git config --global user.email johndoe@example.com

**Your Editor**

* $ git config --global core.editor vim/nano/notepad++

**Checking Your Settings**

* $ git config --list
* $ git config user.name

## **Git Aliases**

* $ git config --global alias.co checkout
* $ git config --global alias.br branch
* $ git config --global alias.ci commit
* $ git config --global alias.st status

## **Cloning an Existing Repository**

* git clone <https://github.com/fvenkat/ReactjsPhotoGallery.git>
  + - * creates ReactjsPhotoGallery dir and it clones
* git clone <https://github.com/fvenkat/ReactjsPhotoGallery.git> <Reactrepo>
* creates Reactrepo dir and it clones

## **Initializing a Git Repository in an Existing Directory**

* $ cd /home/user/my\_project
* $ git init

## **Staging [Tracking new files]**

* $git add . or -A
* $git add <filename>
* $git add -u updated only
* Git add –p

Interactively choose hunks of patch between the index and the work tree and add them to the index. This gives the user a chance to review the difference before adding modified contents to the index

## **Commiting your changes**

* $git commit –m “<commit msg>” filename or **.**
* $git commit –a –m “<commit msg>”

[Skips staging for all **modified files**]

## **Checking the Status of Your Files**

* $ git status
* $ git status –s or git status --short
* **M** README
* **MM** Rakefile
* **A** lib/git.rb [New /untracked file are staged]
* **M** lib/simplegit.rb [Modified /Tracked commited files]
* **??** LICENSE.txt [New files/ untracked]

## **Viewing Your Staged and Unstaged Changes [Git diff]**

**Working Directory VS Staging**

* $ git diff

**Staging vs Local Repo**

* $ git diff --staged or cached
* $git diff --check [identifies possible whitespace]

## **Removing Files**

* $git rm -rvf <filename> [Deletes file from staged + working dir] [changes need to be commited]
* $git rm --cached <filename> [remove file from staging + keeps it in work dir] changes need to be commited]
* $rm –rvf <filename> [remove file from staging + keeps it in work dir] [Changes yet to be staged] [u has use git add . so That file is deleted frm staging as well]

## **Moving Files [Renaming]**

* $ git mv file\_from file\_to

## **Viewing the Commit History [log]**

* $ git log
* $git log –p [which shows the differenceintroduced in each commit]
* $git log –p -2 [shows onle last 2 entry]
* $git log --stat [Prints each commit entry]
* $git log –stat --pretty [changes log output format]
* $git log --oneline
* $git log --oneline –decorate --graph –all
* $ git log --pretty=format:'%h %s' --graph
* $ git log featureA..origin/featureA

[it will print out the history of your commits, showing where your branch pointers are and how your history has diverged]

$ git log --pretty=oneline

$git log --grapgh

$git log --since=2.weeks [until]

$ git log –-merges <<Print only merge commits>>

$ git log --no-merges <<Do not print commits with more than one parent.>>

$ git log --no-merges issue54..origin/master

## **Divergent History**

* $ git log --oneline --decorate --graph –all

## **Undoing Things**

* **Amend**
* $ git commit --amend –m “u can rename the message”
* $ git commit –amend –m “same msg”

<<<this u can update missing file in same msg>>>

At both case hash ID gets renamed

* **Unstaging a Staged File**
  + - $git restore --staged <file>
* **Undo changes in Working tree [Unmodifying a Modified File]**
  + - $git restore <file>

## **Working with Remotes**

**Showing Your Remotes**

* $ git remote
* $ git remote –v [Display remote with url]
* $ git remote show <remote\_name> [Display more info]

**Adding Remote Repositories**

* + $git clone command implicitly adds the origin remote for you
  + $git remote add <remote name> <url>

**Renaming and Removing Remote names**

* + $ git remote rename pb paul
  + $ git remote remove paul

## **Branch Management**

* $ git branch [listing of your local branches]
* $ git branch -r [listing of your remote branches]
* $git branch –v [To see the last commit on each branch]
* $ git branch -vv

*To see what tracking branches*

*If you want totally up to date ahead and behind numbers, you’ll need to fetch from all your remotes right before running this. You could do that like this*

* $ git branch --merged

[To see which branches are already merged into the branch you’re on]

* $ git branch --no-merged

[To see which branches are not merged into the branch you’re on]

## **Creating New Branch**

* $ git branch <List all branches>
* $ git branch <branch name>
* $ git branch testing
* $ git checkout -b <branchname> creates & switch

## **Switching Branches**

* $ git checkout <branch name>
* $ git checkout –b <newbranchname>

## **Delete the branch**

* $ git branch -d hotfix
* $ git branch -D hotfix [force delete]

## **Set upstream /Tracking remote branch**

$ git branch -u origin/serverfix or

$ git branch –set-upstream-to origin/serverfix or

$ git branch –track origin/serverfix

$ git branch –vv

If you want to see what tracking branches you have set up, you can use the -vv option to git branch.

## **Git Fetch**

* $ git fetch <remote name> *Fetch all of the branches from the repository*
* $ git fetch <remote name> <branch name>
* $ git fetch –all *fetches all registered remotes*
* $ git fetch –dry-run
* $git log –oneline master..origin/master
* $git log origin/master
* $ git log --name-status origin/master
* $ git diff master..origin/master
* $ git merge origin/master
* $ git fetch <remote name> <branch name>
* $ git log featureA..origin/featureA
* $ git checkout <branch name>

review & merge

## **Git Pull**

* $ git pull <remote\_name>
* $ git pull –no-commit <remote>
* $ git pull –rebase <remote>

## **Git Push**

* $ git push By default it push the upstream branch [master]
* $ git push <remote name> <branch name>
* $ git push origin master
* $ git push <remote name> <local branch name>:<remote banch name>
* $ git push origin feature1:feature1

*If feature1 not exist in remote then it will create it in remote & then push your changes.*

$ git push origin feature1:master

*Also can be pushed to already existing branch as well.*

## **Deleting Remote Branches**

* $ git push origin --delete serverfix

## **Merging**

**Merging hotfix to master**

* $ git checkout master
* $ git merge hotfix

## **Rebasing**

* $git checkout feature
* $git rebase master
* $git checkout master
* $git merge feature

This moves the entire feature branch to begin on the tip of the master branch

## **Rebase Replay**

* $ git rebase --onto master server client
* $git checkout master
* $git merge client

Take the client branch, figure out the patches since it diverged from the server

branch, and replay these patches in the client branch as if it was based directly off the master branch instead.” It’s a bit complex, but the result is pretty cool.

* $ git rebase master server

This replays your server work on top of your master work

## **Tagging**

**Listing Your Tags**

* $ git tag
* $ git tag –l
* $ $ git tag -l "v1.8.5\*" [particular version]

**Creating Tags**

**Lightweight Tags**

* $ git tag <tagname>
* $ git show <tag name>

**Annotated Tags**

* $ git tag –a <tag name> -m <tag msg>
* $ git tag -a v1.4 -m "my version 1.4"

**Tagging Later**

* **$ git tag –a <tag name> <commit id> -m “tag msg”**

**Sharing Tags**

* $ git push <remote name> <tag name>
* $ git push origin v1.5
* $ git push origin --tags [push all tags]

**Deleting Tags**

* $ git tag -d v1.4-lw
* $ git push origin --delete <tagname>

**Checking out Tags**

* $ git checkout -b version2 v2.0.0
* Switched to a new branch 'version2'

## **Bare repo**

$ git clone --bare <https://github.com/fvenkat/bare.git>

$ git init –bare

**Non bare repo to bare repo**

* $ cd repo
* $ mv .git /desktop/new\_repo # renaming just for clarity
* $ cd ..
* $ rm -fr repo
* $ cd /desktop/new\_repo
* $ git config --bool core.bare true