**Git installation and configuration.(Windows)**

1. Download Git for windows and install with defaults.
2. Install Notepad ++
3. Add environment variables by going to control panel and add Path
4. Configure git by using – Git config --global user.name “name” and

Git config --global user.email “email”

1. Open gitbash and type “notepad++ .bash\_profile”
2. Type “alias npp=’notepad++.exe -multiInst -nosession’” and save file
3. Open git bash and type npp
4. Type pwd (to know present directory)
5. Type “cat ~/.gitconfig” to get configuration details till now.
6. Make notepad as default editor using “git config –global core.editor “notepad++.exe -multiInst -nosession” ”
7. Type “git config --global --list” to list global commands
8. Type “git config --global -e” to edit configuration file.

**Git Commands:**

**Git add Fresh Project:**

1. Mkdir projects (create project folder)
2. Git init fresh-project (initiate git repository)
3. Cd fresh-project (go to repository)
4. Ls (will display files no hidden)
5. Ls -al (will display hidden files also)
6. Cd .git/
7. Ls
8. Cd ..
9. Git status (give what to do next)
10. Touch hipster.txt (create file)
11. npp hipster.txt (Open file)
12. Type some text and save file and close
13. Git status
14. Git add hipster.txt
15. Git status
16. Git commit -m “message”
17. Git status
18. cd ..
19. Rm -rf fresh-project/

**Git usage from existing project.**

1. Download example project from initializr.com
2. Unzip it using unzip “location of file” (unzipping)
3. Ls
4. mv initializr web-project(rename file)
5. ls
6. cd web-project
7. pwd
8. ls
9. git init (initializing git)
10. ls -al
11. git status
12. git add . (adding files to git repository)
13. git commit -m “message” (commit git)
14. git status
15. ls -al
16. rm -rf .git (files are no more managed by git)
17. ls -al
18. cd ..
19. rm -rf web-project (remove project)
20. ls

**Pushing to github**

1. git clone <github url> (clone github url)
2. ls
3. cd starter-web
4. ls
5. ls -al
6. git status
7. git status
8. touch hipster.txt
9. npp hipster.txt
10. ls
11. git status
12. git add hipster.txt
13. git status
14. git commit -m “my first commit”
15. git status
16. git pull origin master
17. git push origin master
18. git status

**Tracked files**

1. ls
2. npp hipster.txt and add some more text
3. git status
4. git commit -am “modified file” (adding and committing at a once)
5. git status
6. git ls-files(gives list of tracked files)
7. npp newfile.txt
8. ls
9. git status
10. git ls-files
11. git add newfile.txt
12. git ls-files
13. git commit -m “message”

**Recursive Add**

1. mkdir -p level1/level2/level3
2. ls
3. cd level1
4. npp level1text.txt
5. ls
6. cd level2
7. npp level2text.txt
8. ls
9. cd level3
10. npp level3text.txt
11. cd ..
12. cd ..
13. cd ..
14. git add .
15. git status
16. git commit -m “message”

**Backing Out Changes**

1. cd level1
2. npp level1file.txt
3. add some text and save file
4. git status
5. git add level1file.txt
6. git status
7. git restore --staged level1file.txt (get back to unstagged area)
8. git status
9. git restore levelfile.txt (remove text from working directory)
10. git status.

**Rename and Move files.**

1. Cd level1/level2/level3
2. Ls
3. Git status
4. Git mv level3file.txt level3.txt (rename using git)
5. Ls
6. Git status
7. Git commit -m “renaming level3file”
8. Git status
9. Cd ..
10. Ls
11. Mv level2file.txt level2.txt(renaming using cmd)
12. Ls
13. Git status
14. Git add -A
15. Git commit -m “moving using command prompt instead of git”
16. Git status
17. Ls
18. Git mv level2.txt 2.txt
19. Ls
20. Git status
21. Git mv 2.txt level2.txt
22. Ls
23. Git status
24. Ls
25. Git mv level2.txt level3(moving from level2 to 3)
26. Ls
27. Cd level3
28. Ls
29. Git status
30. git commit -m “moving file from level2 to 3”
31. ls
32. mv level2.txt .. (moving back to level2)
33. ls
34. git status
35. git add -A
36. git status
37. git commit -m “moving level2 file back to level2”

**Deleting Files**

1. npp doomed.txt and type some text
2. ls
3. git status
4. git rm doomed.txt (gives error didn’t match filer as it is not tracked by git)
5. rm doomed.txt
6. git status
7. ls
8. git ls-files
9. git rm newfile.txt(remove file using git)
10. git status
11. git commit -m “new file deleted”
12. git rm hipster.txt
13. ls
14. git status
15. git restore --staged hipster.txt (restoring back to staging area)
16. ls
17. git status
18. git restore hipster.txt (restore file completed to work directory)
19. git ls-files
20. rm hipster.txt
21. git status
22. git add -A (adding/deleting files from workdirectory to git)
23. git status
24. git commit -m “deleted hipster.txt”
25. git status
26. rm -rf level1 (remove all folder structure)
27. git status
28. git add -A
29. git status
30. git commit -m “deleting level1 in all children”

**Log**

1. git help log (help on log file)
2. git log (gives commit history)
3. git log --abbrev-commit (shorten commit history)
4. git log --oneline --graph –decorate (give all comits in one line)
5. git log <commitid start>…<commitid end>
6. git log --since=”3 days ago”
7. git log --follow -- level level1/level2/level2.txt(gives history os specific file with renamed)
8. git show <commit id> (show details of commit)
9. git log -- hipster.txt (gives commit history of specific file)

**Alias**

1. git status
2. git config --global alias.hist “log --all --graph --decorate --oneline”
3. git hist
4. npp ~/.gitconfig
5. Change command in hist to “log --all --decorate --oneline”
6. Git hist
7. npp ~/.gitconfig
8. Change back command in hist to “log --all --graph --decorate --oneline”
9. Git hist

**Git ignore**

1. npp .gitignore
2. git status
3. git add .gitignore
4. git commit -m “adding gitignore”
5. npp access.log
6. npp .gitignore and add content “\*.log” (ignore all files with .log)
7. git status
8. mkdir log
9. mv access.log log
10. cp access.log access.2020
11. git status
12. npp .gitignore and add “log/” (ignore log directory)
13. git status.

**P4Merge**

1. Download p4merge and install.
2. Open git bash and type below commands
3. Git config --global merge.tool p4merge
4. Git config --global mergetool.p4merge
5. Git config --global mergetool.p4merge.path “c:/program files/perforce/p4merge.exe”
6. Git config --global mergetool.prompt false
7. Git config --global diff.tool p4merge
8. Git config --global difftool.p4merge
9. Git config –global difftool.p4merge.path “c:/program files/perforce/p4merge.exe”
10. Git config –global difftool.prompt false

**Diff tool**

1. Make some changes to readme in working dir
2. Go to git bash and type “git difftool” (compare between working directory and staging area)
3. Git diff HEAD
4. Git difftool HEAD(compare between work directory and git repository)
5. Git diff --staged HEAD
6. Git difftool --staged HEAD (compare between staging area and git repository)
7. Git difftool – README.md (to show only changes in one file)
8. Git log --oneline
9. Git diff <commitid> HEAD (compare between specific git id and last commit)
10. Git difftool <commitid> HEAD
11. Git diff HEAD HEAD^ (compare last and n-1 commit ids)
12. Git difftool HEAD HEAD^
13. Git diff <commitid1> <commitid2> (compares two commit ids)
14. Git difftool <commitid1> <commitid2>
15. Git diff master master/origin (compare between local and remote repository)
16. Git difftool master master/origin

**Branching**

1. Git branch -a (display current branches available)
2. Git branch mynewbranch (create new branch)
3. Git branch -a
4. Git checkout mynewbranch (switch to new branch/ point Head on it)
5. Git branch -a
6. Git log --oneline --decorate
7. Git checkout master
8. Git branch -a
9. Git branch -m mynewbranch newbranch(rename branch)
10. Git branch -a
11. Git branch -d newbranch(delete branch)
12. Git branch -a

**Merging (Fast forward merge)**

1. Git status
2. Git branch (show current branches only local)
3. Git checkout -b title-change(create branch and make it as HEAD)
4. Git status
5. “npp simple.html” and change title in html page
6. Git status
7. Git commit -am “changing title of html page”
8. Git log --oneline
9. Git checkout master
10. Git difftool master title-change
11. Git merge title-change(Fast forward merge as we havn’t changed any thing in master)
12. Git status
13. Git log --oneline --graph –decorate
14. Git branch -d title-change (delete branch)
15. Git branch -a

**Merging (no Fast forward merge)**

1. Git status
2. Git branch
3. Git checkout -b add-copyright
4. Git branch
5. npp simple.html
6. add content.
7. Git status
8. Git commit -am “adding copyright”
9. npp readme.md
10. add content
11. git status
12. git commit -am “adding copyright to readme”
13. git log --oneline --graph --decorate
14. git checkout master
15. git merge add-copyright --no-off (no fast forward merge ask for message to disaplay)
16. git log --oneline --graph --decorate
17. git branch -d add-copyright

**Automatic Merges**

1. git status
2. git branch
3. git checkout -b simple-changes
4. npp humans.txt and edit content
5. git status
6. git commit -am “adding team members”
7. git checkout master
8. npp readme.md
9. git status
10. git commit -am “adding instructions on how to contribute”
11. git log --oneline --graph --decorate --all
12. git branch
13. git merge simple-changes -m “merging changes from simple-chnages”
14. git log --oneline --graph --decorate --all
15. git branch -d simple-changes

**Resolving Merge conflicts**

1. git status
2. git branch
3. git checkout -b realwork
4. npp simple.html and make come changes
5. git status
6. git commit -am “making changes to simple.html”
7. git status
8. git checkout master
9. npp simple.html and make come changes
10. git add simple.html
11. git commit -m “making conflicts”
12. git status
13. git log --oneline --graph --decorate --all
14. git branch
15. git merge realwork (gives error automatic mege fail)
16. “git mergetool” and resolve conflicts
17. Git commit -m “done resolving conflicts”
18. Git status
19. npp .gitignore
20. add \*.orig
21. git add .gitignore
22. git commit -m “adding orig to git ignore”
23. git branch -d realwork
24. git log --oneline --graph --decorate --all
25. git status

**Git Rebase**

1. git checkout -b myfeature
2. npp humans.txt and add some text
3. git status
4. git commit -am “saying thanks to all students”
5. git status
6. git checkout master
7. npp readme.md and made some changes.
8. Git status
9. Git commit -am “adding text to readme”
10. Git log --oneline --decorate --all –graph
11. Git checkout myfeature
12. Git rebase master
13. Git status
14. Git log --oneline --decorate --all –graph
15. Npp readme.md and add some lines
16. Git status
17. Git commit -am “adding another change after rebase”
18. Git log --oneline --decorate --all –graph
19. Git checkout master
20. Git difftool master feature
21. Git merge myfeature
22. Git status
23. Git log --oneline --decorate --all –graph
24. Git branch -d myfeature
25. Git log --oneline --decorate --all –graph

**Rebase commands**

1. Git rebase --abort (to stop rebase while in conflicts)
2. Git rebase --continue(to continue rebase after resolving conflicts)

**Pull with Rebase(Github)**

1. Git checkout master
2. Git pull origin master
3. Git push origin master
4. Npp simple.html and make some changes
5. Git commit -am “message”
6. Git status
7. Go to git hub and make sum changes to index.html
8. Git fetch origin master
9. Git status
10. Git pull --rebase origin master
11. Git status
12. Git log --oneline --graph --decorate –all

**Git stashing** (will pause the working file and come back after completing editing some priority file)

1. Pwd
2. Git status
3. Ls
4. Npp simple.html and edit some text
5. Git status
6. Git stash
7. Git status
8. Npp simple.html (show unchanged file)
9. Npp readme.md and edit some text
10. Git commit -am “quick fix in production”
11. Git status
12. Git stash apply (back to simple.html file)
13. Npp simple.html (show edited content)
14. Git status
15. Git commit -am “done with simple.html”
16. Git status
17. Git stash list (will give list of work in progress files)
18. Git stash drop (need to drop after completing your work with simple.html)
19. Git stash list

**Stashing untracked files**

1. Git ls-files
2. Npp humans.txt and edit some text
3. Git status
4. Npp newfile.txt (untracked file)
5. Git status
6. Git stash
7. Git stash apply
8. Git stash drop
9. Git stash list
10. Git stash -u
11. Git status
12. Git stash list
13. Npp readme.md and edit some text
14. Git commit -am “quick fix to readme after commiting”
15. Git status
16. Git stash apply
17. Git stash drop
18. Git stash
19. Git stash pop (apply + drop)
20. Git status
21. Rm newfile.txt
22. Git status
23. Git commit -am “update to human files after stash pop”
24. Git status

**Manage multiple stash**

1. Ls
2. Npp simple.html and made some changes
3. Git stash save “simple chnages” (stash with message)
4. Npp index.html and made changes
5. Git stash save “index changes”
6. Npp readme.md
7. Git stash save “readme changes”
8. Git stash list
9. Git stash show stash@{1} (show specific stash)
10. Git status
11. Git stash list
12. Git stash apply stash@{1} (apply stash to specific file)
13. Git stash list
14. Git stash drop stash@{1} (drop specific stash file)
15. Git stash list
16. Git stash clear (drop all stashes)
17. Git stash list
18. Git status

**Stashing into a branch**

1. Git status
2. Git stash list
3. Npp simple.html and add some text
4. Npp humans.txt and add some text
5. Git status
6. Git add index.html
7. Npp new.md
8. Git status (has untracked, tracked and changes to be committed files)
9. Git stash -u
10. Git status
11. Git stash branch newchanges (will create new branch and stash and drop)
12. Git status list
13. Git status
14. Rm new.md
15. Git add .
16. Git status
17. Git commit -m “changes after git bash commit”
18. Git checkout master
19. Git merge newchanges.
20. Git status
21. Git branch -d newchnages
22. Git branch

**Simple Tag/Light Weight Tag.**

1. Git log –oneline –graph –decorate –all
2. Git tag myTag (Light weight tag)
3. Git log –oneline –graph –decorate –all
4. Git tag –list
5. Git show mytag
6. Git tag --delete mytag
7. Git tag --list.

**Annotated tag (giving tag with message)**

1. Git tag -a v-1.0(annotated tag)
2. It will open text editor, type release 1.0
3. Git tag --list
4. Git log –oneline –graph –decorate –all
5. Git show v-1.0

**Comparing Tags**

1. Git tag --list
2. Git ls
3. Npp index.html and edit some text
4. Git status
5. Git commit -am “tweaking file for tagging example”
6. Git log –oneline –decorate –graph –all
7. Git tag -a v-1.1
8. Npp simple.html and edit file
9. Git commit -am “updating for tag 1.1” (updating wrongly tag 1.1 instaed of 1.2)
10. Git commit --amend (revert the commit by amend and change)
11. Git tag v-1.2 -m “Release 1.2” (other way to annotate a tag)
12. Git tag --list
13. Git diff v-1.0 v-1.2 (compare tags)

**Tagging specific commit**

1. Git log –oneline –decorate –graph –all
2. Copy any one of the commit id
3. Git tag -a v-0.9-beta <commit id>
4. Git log –oneline –decorate –graph –all

**Updating Tag**

1. Git log –oneline –decorate –graph –all
2. Git tag -a v-0.9-beta -f <new commit id to update tag>
3. Git log –oneline –decorate –graph –all

**Tag with Github**

1. Go to github -> releases-> tags and see list of tags in github.
2. Git tag --list.
3. Git log –oneline –decorate –graph –all
4. Git push origin v-0.9-beta (will push specific tag)
5. Git push origin master --tags (push all tags).
6. Git tag --list.
7. Git push origin :v-1.0 (delete specific tag)

Git cherry pick

1. Git cherry-pick <commit id> is used to shift commit from one branch to another