**DevOps Foundations for Java**

**with**

**Git, Jenkins, and Maven**

**kiran**

**Table of Contents**

[Lab 1 - Version Control - Git 3](#_TOC_250002)

[Lab 2 - Branching, Merging and Working with Remotes 8](#_TOC_250001)

[Lab 3 - GitFlow Workflow 14](#_TOC_250000)

Lab 4 - Configure Tools in Jenkins 24

Lab 5 - Install Maven 29

Lab 6 - Create a Maven Project 34

Lab 7 - Create a Web Application from an Archetype 41

Lab 8 - Add Web Site, Static Analysis, and Code Coverage 53

Lab 9 - Create a Jenkins Job 62

Lab 10 - A Multi-Module Project 80

Lab 11 - Create a Standard Set of Dependencies 89

Lab 12 - Write a Simple Maven Plugin 92

Lab 13 - Create an Archetype 103

Lab 14 - Create a Corporate Repository with Artifactory 112

Lab 15 - Deploy to the Corporate Repository with Artifactory 118

Lab 16 - Add Development Metrics 124

Lab 17 - Create a Pipeline 131

# Lab 1 - Version Control - Git

In this chapter you will install, configure, and use Git.

### Part 1 - Launch terminal

In this part you will launch the Linux terminal.

1. Open the **Terminal** window.

Alternatively, you can press Ctrl+Alt+T to access the terminal.

### Part 2 - Install Git

1. Update APT repository

sudo apt-get update

Enter wasadmin as password, if prompted

2. Install git.

sudo apt-get install git

Press Y, if prompted

3. In the terminal run following command to find git version number.

git –version git help

git help -a

### Part 3 - Using Git

In this part you will use Git to perform various operations, such as, check in, check status etc.

1. Switch to the "Documents" directory.

cd ~/Documents

2. Create a directory.

mkdir -p workspace/git

3. Switch to the "git" directory.

cd workspace/git

4. Initialize repository.

git init

5. Tell Git who you are.

git config --global user.name "Alice Smith" git config --global user.email [alice@smith.com](mailto:alice@smith.com)

Note: One interesting aspect of Git is that it separates user identity in the repository from any sort of authentication or authorization. Because a distributed repository will generally be maintained by many separate individuals or systems, the identity of the committer must be contained in the repository – it can't just be supplied as a user id when we do the commit. So, even if we're not connected to any central repository, we need to tell Git who we are. The identity that we supply will be recorded whenever we commit to a repository.

6. Create a text file.

nano sample.txt

7. Enter following text.

First Version!

8. Press Ctrl+O to save the file and hit enter.

9. Press Ctrl+X to exit to the terminal.

10. Get Git status.

git status

Notice sample.txt is listed under untracked files.

11. Add the files to tracked.

git add .

Note: Here you are adding the current directory. You could also add the file using "git sample.txt".

12. Get Git status again.

git status

Notice sample.txt is tracked.

13. Commit changes.

git commit

Notice it launched text editor automatically which lists operations that will get performed when files are committed. Here you can add detailed description that will get saved when you commit the changes.

14. Add following text in the first line.

Added sample.txt

15. Press Ctrl+O to save the file and hit enter.

16. Press Ctrl+X to exit to the terminal.

17. Get Git status.

git status

Notice it says there's nothing to commit since you have already committed all changes.

18. Modify sample.txt.

nano sample.txt

19. Change "First Version!" to "Second Version!"

20. Press Ctrl+O to save the file and hit enter.

21. Press Ctrl+X to exit to the terminal.

22. Get Git status.

git status

Notice it says the file is modified.

23. View changes.

git diff

Notice it shows old text in red and new text in green.

24. Create another file.

nano another.txt

25. Add the following text.

Hello World!

26. Press Ctrl+O to save the file and hit enter.

27. Press Ctrl+X to exit to the terminal.

28. Add all files.

git add .

29. Get Git status.

git status

Notice it's showing 1 file as modified and 1 file as a newly added file.

30. Commit changes.

git commit -m "Made 2 changes"

Notice when you pass -m switch, you can store a simple single line comment.

31. Get Git status.

git status

Notice there's nothing to commit.

32. Delete sample.txt

rm sample.txt

33. Recover file.

git checkout sample.txt

34. View sample.txt

cat sample.txt

Note: It restored latest version by default.

35. Delete file again.

rm sample.txt

36. View all versions of a file.

git log

Notice it shows user, commit id, date time, and comment.

37. Copy the **commit id** for the older version.

38. View changes between current and the first version.

git diff <commit\_id>

39. Restore the older version.

git checkout <commit\_id> sample.txt

40. View file content.

cat sample.txt

Notice it's the first version.

41. Close the terminal.

### Part 4 - Review

In this chapter you installed and used Git.

# Lab 2 - Branching, Merging and Working with Remotes

One of the main themes of Git is the idea that we should isolate work on branches and use those branches to coordinate the work of multiple developers. In this lab, we'll examine the use of branches and the act of merging.

In addition, Git works on the basis of distributed repositories. We will explore the use of more than one repository, or Remote repositories.

**At the end of this lab you will be able to:**

1. Clone an existing repository
2. Create a branch
3. Merge a branch onto another branch.

### Part 1 - Launch terminal

In this part you will launch the Linux terminal.

1. Open the **Terminal** window.

Alternatively, you can press Ctrl+Alt+T to access the terminal.

### Part 2 - Clone a Repository

For this lab, we will clone a pre-existing repository that has been provided to you.

1. Switch to the "Documents" directory.

cd ~/Documents

2. Switch to the "git" directory.

cd workspace/git

3. Clone the repository:

git clone ~/Downloads/BranchMergeRemotes

This command creates a clone of the repository that is located in the file system.

In this case, we're cloning from a repository that is in the file system, but the exact same syntax can be used to clone from a repository on a remote server. The only difference is the URL that's used to identify the original repository.

For instance, 'git clone https://github.com/angular/angular.js.git' would create a local copy of the repository that contains the popular 'AngularJS' framework, which is hosted on GitHub.

4. In the **Terminal** window, type the following:

cd BranchMergeRemotes

### Part 3 - Create a Branch to Work on a Feature.

In many version control systems, particularly older systems, creating a branch is a difficult and often expensive operation. In Git, however, creating a branch is a very lightweight operation, and in general, we want to create a branch for every feature that we work on. When a feature reaches a stable state, we'll merge it back to the master branch. In this way, the master branch only contains stable features, and any developer's work is isolated in its own feature branch.

1. In the **Terminal** window, type the following, pressing return after each line:

git branch add-text git checkout add-text

The lines above create a new branch called 'add-text' (which is a description of what we are about to do), and then switch to that branch.

Note: The two commands above could have been shortened to 'git checkout -b add-text'.

2. Get Git status.

git status

Notice that the status message indicates that we are on the branch called 'add-text'. That's all that's required to create a branch in Git!

3. Enter the branch command:

git branch

The branch command on its own simply lists the branches that are defined in the repository. Notice that the 'add-text' branch is both highlighted and has an asterisk next to its name, indicating that we are on that branch.

### Part 4 - Work on the Branch

When we are on a branch, we can go ahead and make some changes and then stage and commit those changes.

1. Open the index file.

nano index.html

Enter wasadmin or the password provided by your organization. Contact your instructor to get this information.

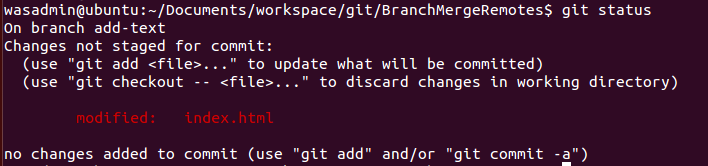
2. Replace the text 'Hello World.' with 'This is the text that we really want.'

3. Press Ctrl+O to save the file and hit enter.

4. Press Ctrl+X to exit to the terminal.

5. Get Git status.

git status



Notice that git now tells us that the file '**index.html**' has been updated, but not staged. We could use the 'git add' command to stage the changed file, but instead we are going to add an option to 'git commit' that will automatically stage all the changes.

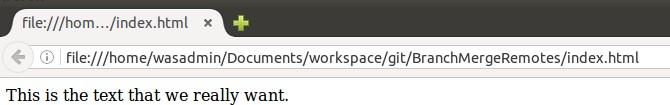
6. Type the following and then press return:

git commit -a -m "We now have the text we want."

7. Open a file browser and navigate to the following folder.

~/Documents/workspace/git/BranchMergeRemotes/

8. Right click on '**index.html**' and select 'Open with Firefox Web Browser'. Notice that the newer text is displayed.



9. Close the web browser.

### Part 5 - Merge Back to the Master Branch

We have committed changes to the feature branch that we created and called 'add-text'. Now let's say that we are happy with the feature and want to 'promote' it to the master branch.

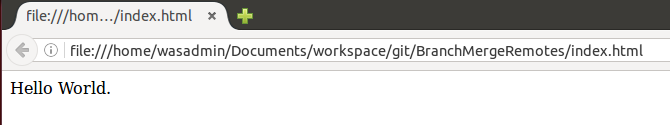
The general procedure is as follows:

* Switch to the master branch
* Merge the changes from the feature branch
* Optionally, delete the feature branch

1. In the **Terminal** window, enter the following line:

git checkout master

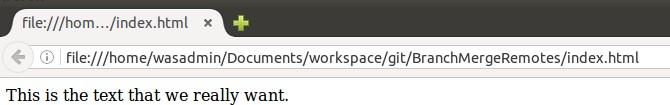
2. As above, open '**index.html**' with a web browser, but notice that the text now reads 'Hello World' again.



3. Enter:

git merge add-text

4. As above, open '**index.html**' with a web browser, but notice that we have the updated text.



5. We are done with this feature branch, so let's delete it.

git branch -d add-text

There's no cost the keeping the branch around (remember that a branch is really just a reference to a particular committed snapshot), but it will generally prevent confusion if we keep the branch set clean by deleting branches that we are not actively using.

### Part 6 - Push to Another Remote Repository

Let's say we wanted to create a new repository and store all our changes there. In essence, this is what we would do when we first create a project.

As it stands, we have a local repository that contains a number of snapshots. The basic procedure to create a remote repository will be:

* Create an empty remote repository
* Push our changes to that repository

For simplicity, we will use another repository in the local file system, but as with pulling

from a remote server, the only difference would be the url used for the repository.

1. Create a new folder.

mkdir ~/Documents/workspace/AnotherRemote

2. Make sure you are on the BranchMergeRemotes.

cd ~/Documents/workspace/git/BranchMergeRemotes/

3. In the **Terminal** window, type the following:

git init --bare ~/Documents/workspace/AnotherRemote

The '--bare' option tells git that this repository will never have a working copy. It will only be used as a git repository. This is usually the case for 'remote' repositories that are used to share code. Setting up the repository in this way allows us to "push" changes to it freely, even if our changes would make a working directory obsolete if it existed.

Otherwise, we would need to go into that repository and "pull" changes into it.

If you have a look at the contents of **\workspace\AnotherRemote**, you will notice that it's different from the repositories we've already seen – there is no hidden '**.git**' folder. Rather, the contents that would have been in that folder are present in the **AnotherRemote** folder.

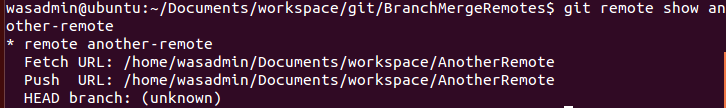
4. In the **Terminal** window, type the following:

git remote add another-remote ~/Documents/workspace/AnotherRemote

This command tells git to add a reference to a remote repository at '/workspace/AnotherRemote', and to call it 'another-remote' locally.

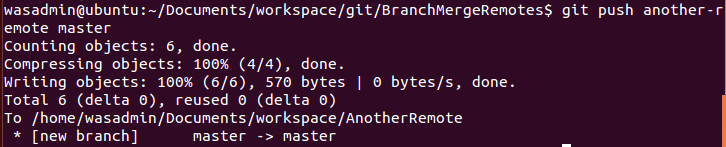
5. We can display the remote repository's status.

git remote show another-remote



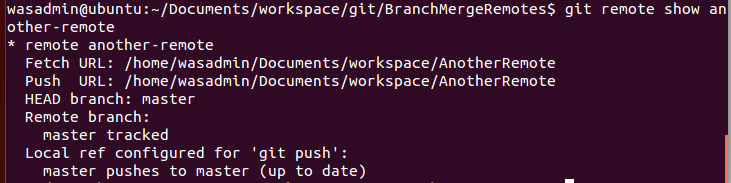
6. Finally, let's push our data to the remote.

git push another-remote master



7. Display the remote repository's status.

git remote show another-remote



Notice that this time, the remote status indicates that the remote HEAD branch is 'master' and that we are tracking the remote master branch.

8. Close the Terminal window and browsers.

### Part 7 - Review

In this lab, we tried out a common workflow, the "feature branch" workflow. We checked out a project stored in a remote repository. We created a new branch, did some work on it, committed it, and then merged it back onto the master branch. Finally, we created yet another remote repository and pushed our changes to it.

The thing to notice here is that using remote repositories is natural in git. In fact, the "remote" repository is basically identical to the "local" repository – in this case, we used simple repositories in the file system, but they could equally have been accessed remotely using secure-shell or a repository manager.

# Lab 3 - GitFlow Workflow

## .

In this lab you will explore GitFlow feature, release, hotfix, and various other options.

### Part 1 - Create a directory for storing GitFlow lab content.

In this part you will create a directory for storing GitFlow lab content.

1. Open a command prompt window.

2. Create your working directory.

md c:\workspace

3. Change to your working directory.

cd \workspace

4. Create a directory for storing GitFlow lab content.

mkdir gitflow\_test

5. Switch to the created directory.

cd gitflow\_test

### Part 2 - Create and setup Remote and Local repositories.

In this part you will create a remote and a local repository.

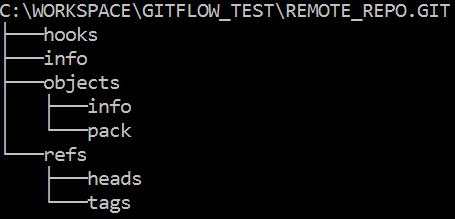
1. Initialize a new bare repository.

git init --bare remote\_repo.git

2. Verify the remote repository is created.

tree remote\_repo.git

Notice it shows directory tree like this.



3. Clone the bare remote repository to a local working copy.

git clone remote\_repo.git local\_repo

Notice it's a blank repository that's why there is warning displayed on the screen.

4. Verify local repository is created.

tree local\_repo

Notice the directory structure is blank.

5. Switch to the local working copy.

cd local\_repo

6. Configure user email for the repository.

git config user.email ["bob@abc.com"](mailto:bob@abc.com)

7. Configure user name for the repository.

git config user.name "Bob"

### Part 3 - Set up Gitflow in the local repository

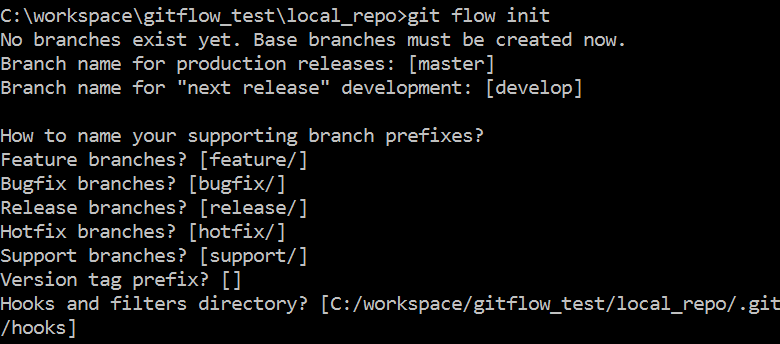
In this part you will set up Gitflow in the local Repository. You will create various branches required by the Gitflow.

1. Initialize Gitflow in the local repository.

git flow init

2. Press the **Enter** key to use default values for all options.

Notice the screen looks like this.



3. View the branch list.

git branch



Notice it shows following branches.

"develop" is the active branch. In case if you want to activate a different branch, use following command:

git checkout <branch>

### Part 4 - Create a Gitflow feature.

In this part you will create a Gitflow feature.

1. Get feature list.

git flow feature list

Notice currently there are no features.

2. Create a new feature.

git flow feature start HOME\_PAGE

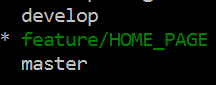
3. Get feature list again.

git flow feature list

Notice it shows HOME\_PAGE feature.

4. Get branch list.

git branch



Notice it shows following branches:

feature/HOME\_PAGE feature has been added as a branch.

5. Get git status.

git status

Notice feature/HOME\_PAGE is the active branch and currently there are no files to commit.

### Part 5 - Add content as part of the feature.

In this part you will create an HTML file as part of the feature created in the previous part.

1. Start Notepad.

notepad index.html

2. Click Yes to create the file.

3. In the Notepad window enter following text.

<html>

<body>

</body>

</html>

4. Save the file and close the Notepad window.

5. Get git status.

git status

Notice "index.html" file isn't available in the repository.

6. Add index.html to the local repository's feature/HOME\_PAGE branch.

git add index.html

7. Get git status.

git status

Notice index.html is added the repository but it's not committed yet.

8. Commit changes.

git commit -m "Added index.html"

9. View **feature/HOME\_PAGE** branch contents.

git ls-tree -r --name-only feature/HOME\_PAGE

Notice it shows index.html. If you don't use --name-only switch, it will show file size in addition to the file ID.

10. Merge **feature** branch into the **develop** branch without deleting the feature branch.

git flow feature finish HOME\_PAGE --no-ff -k

Notice feature/HOME\_PAGE branch has been merged into the develop branch. The -k switch keeps the feature branch intact. Without this switch, it will delete the feature branch. Skip -k switch only when the feature has been fully developed and you want to merge the feature into the develop branch and then delete the feature branch. Also note that you are switched to the develop branch after performing the merge operation.

The –no-ff switch disables "fast forward".

11. View **develop** branch contents.

git ls-tree -r --name-only develop

12. Switch back to the feature branch.

git checkout feature/HOME\_PAGE

13. Open index.html file in notepad.

notepad index.html

14. Add <h1> tag to the **body** tag.

<h1>Hello world!</h1>

15. Save the file and close the Notepad window.

16. Get git status.

git status

Notice index.html file is updated on the file system, but it's not updated in the repository.

Also notice you are on the develop branch.

17. Stage the changed index.html.

git add index.html

18. Commit changes to the feature branch.

git commit -m "Added h1 tag to the index.html"

19. Merge **feature** branch into the **develop** branch without deleting the feature branch.

git flow feature finish HOME\_PAGE --no-ff -k

20. View remote repository content.

git ls-remote

Notice currently no branches are in the remote repository. All the changes are in the local repository right now.

21. Push changes from local repository into the remote repository.

git push --all

22. View remote repository content.

git ls-remote

Notice it displays **develop** and **master** branches.

### Part 6 - Create a new Release

In this part you will create Gitflow release.

1. Create a new release "REL\_1.0" based on **develop** branch.

git flow release start 'REL\_1.0' develop

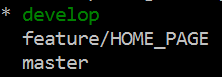
2. Finish the release REL\_1.0 and delete the release branch.

git flow release finish 'REL\_1.0' -m "v1.0"

Notice release branch **REL\_1.0** has been merged into master. The release has been back- merged into develop. You are switched to the **develop** branch.

3. Get branch list.

git branch



Notice it shows branch list like this.

4. Push all changes to the remote repository.

git push --all

5. Switch to the feature/HOME\_PAGE branch.

git checkout feature/HOME\_PAGE

6. Open index.html file in Notepad.

notepad index.html

7. Below h1 tag, add following text.

Welcome to ABC Inc.

Notice you haven't enclosed the above text in any tag. You are purposefully skipping a tag. You will treat as a bug and fix it later.

8. Save the file and close the Notepad window.

9. Stage the changed index.html.

git add index.html

10. Commit the changes.

git commit -m "Added company name to index.html"

11. Merge feature changes into the **develop** branch without deleting the feature branch.

git flow feature finish HOME\_PAGE --no-ff -k

12. Start another release from **develop** branch.

git flow release start 'REL\_2.0'

13. Finish the release REL\_2.0 and delete the release branch.

git flow release finish 'REL\_2.0' -m "v2.0"

Notice release REL\_2.0 has been merged into 'master'. The release was tagged 'REL\_2.0'. Release REL\_2.0 has been back-merged into 'develop'. Release branch REL\_2.0 has been locally deleted and you are switched to 'develop' branch.

14. View tag list.

git tag

Notice it shows REL\_1.0 and REL\_2.0.

If you want to checkout a certain release based on tag, you can execute following command:

git checkout tags/<tag\_name>

### Part 7 - Working with Hotfixes.

In this part you will create a Gitflow hotfix.

1. Create a new hotfix.

git flow hotfix start 'REL\_2.0.1'

Notice hotfix will end up creating a new release.

2. Open index.html file in the Notepad.

notepad index.html

3. Enclose Welcome to ABC Inc. within <b> HTML tag.

4. Save the file and close the Notepad window.

5. Stage the changed index.html file.

git add index.html

6. Commit changes.

git commit -m "Enclosed company name in the bold HTML tag"

7. Finish the hotfix.

git flow hotfix finish 'REL\_2.0.1' -m "v2.0.1"

Notice the hotfix branch has been merged into 'master'. The hotfix was tagged REL\_2.0.1. The hotfix has been back-merged into 'develop' branch and has been locally deleted. If you want to keep the hotfix branch then use -k switch. You are switched to the 'develop' branch.

8. Push all changes into the remote repository.

git push --all

### Part 8 - Verify all releases are created

In this part you will verify all releases are created.

1. Get tag list.

git tag

Notice REL\_1.0, REL\_2.0, and REL\_2.0.1 are available.

2. View current tag name.

git describe

Notice the prefix is REL\_2.0.1

3. Checkout REL\_1.0

git checkout tags/REL\_1.0

4. View current tag name.

git describe

Notice the prefix is REL\_1.0

5. View index.html file contents.

type index.html

Notice there's no "Welcome to ABC Inc." text.

6. Checkout REL\_2.0.

git checkout tags/REL\_2.0

7. View current tag name.

git describe

Notice the prefix is REL\_2.0.

8. View index.html file contents.

type index.html

Notice there's "Welcome to ABC Inc." text, but it's not enclosed in <b> tag.

9. Checkout REL\_2.0.1.

git checkout tags/REL\_2.0.1

10. View current tag name.

git describe

Notice the prefix is REL\_2.0.1.

11. View index.html file contents.

type index.html

Notice there's "Welcome to ABC Inc." text enclosed in <b> tag.

12. Close all.

### Part 9 - Review

In this lab you explored Gitflow feature, release, hotfix, and various other options.