GIT
VERSION CONTROL

BOSCH

Git

AGENDA

- Introduction
- SDLC Models
- Installation and Tools for Git
- Sub Version & Git
- Git Snapshots
- Basic Git Workflow
- Initial Git Configuration
- Creating Git Repo
- Git Commands
- Git Network
- Git Best Practices
- References to Git

Git
Introduction

Created by Linus Torvalds,

- creator of Linux, in 2005
- Came out of Linux development community
- Designed to do version control on Linux kernel

(A "git" is a cranky old man. Linus meant himself.)

Goals of Git:

- Speed
- Support for non-linear development
  (thousands of parallel branches)
- Fully distributed
- Able to handle large projects efficiently

Git
Installation and Tools for Git

Git website: http://git-scm.com/

Tortoise Git: https://tortoisegit.org/download/

Source Tree: https://www.sourcetreeapp.com/

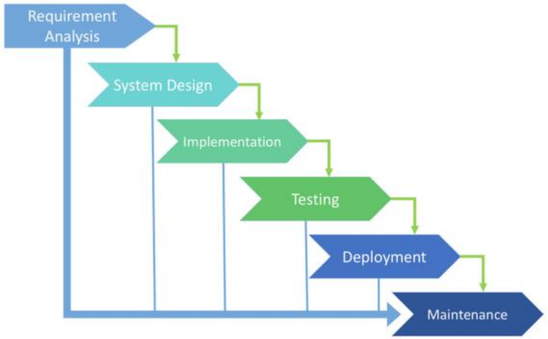
Other Tools:

- Aurees
- Gitg
- Qgit

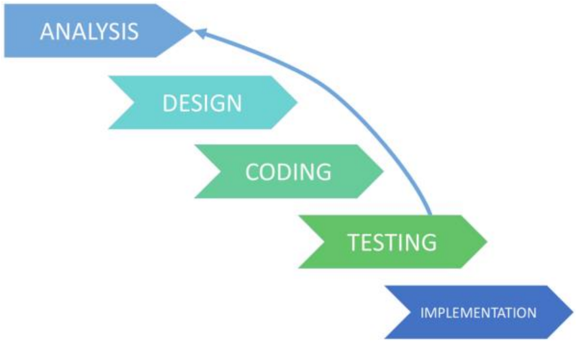
- At command line:
  - *`git --version`*
  - *`git help verb`*

    (where verb = config, add, commit, etc.)

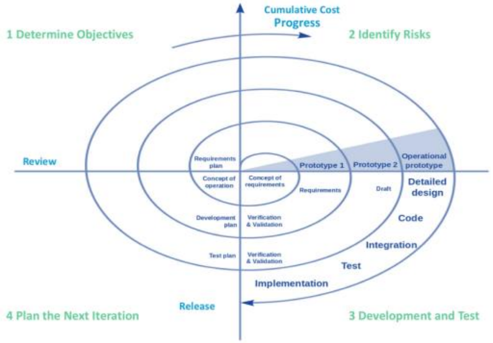
Git
SDLC Modules



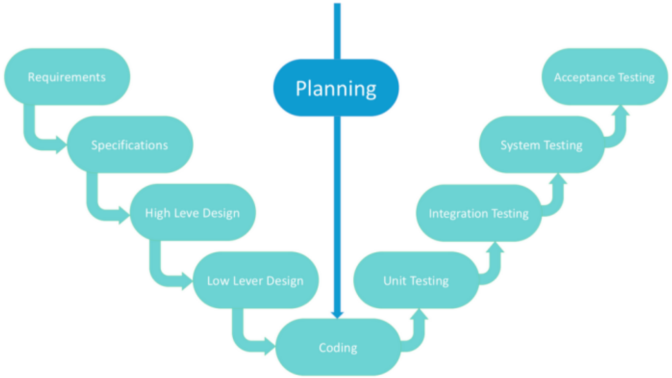
Waterfall Model



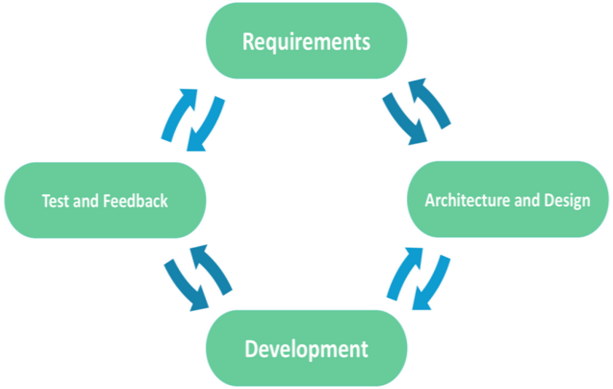
Iterative Model



Spiral Model



V-shaped Model



Agile Model

Git
Centralized VCS

- In Subversion, CVS, Perforce, etc.

  A central server repository (repo) holds the "official copy" of the code

  – the server maintains the sole version history of the repo

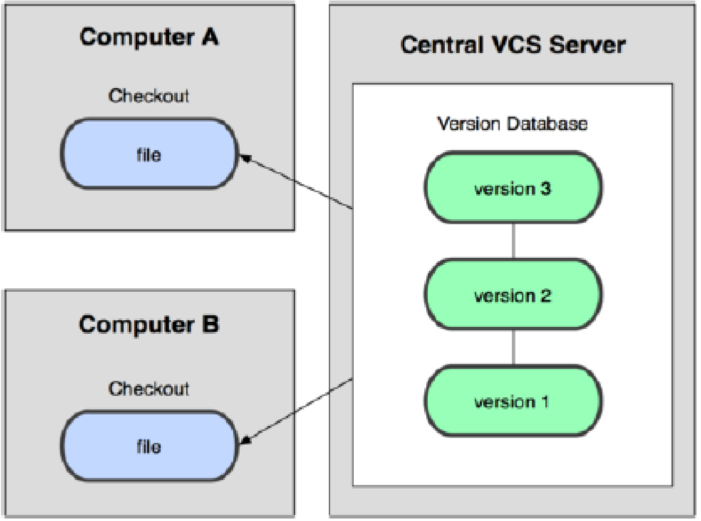
- You make "checkouts" of it to your local copy

  – you make local modifications

  – your changes are not versioned

- When you're done, you "check in" back to the server

  – your checkin increments the repo's version



Git

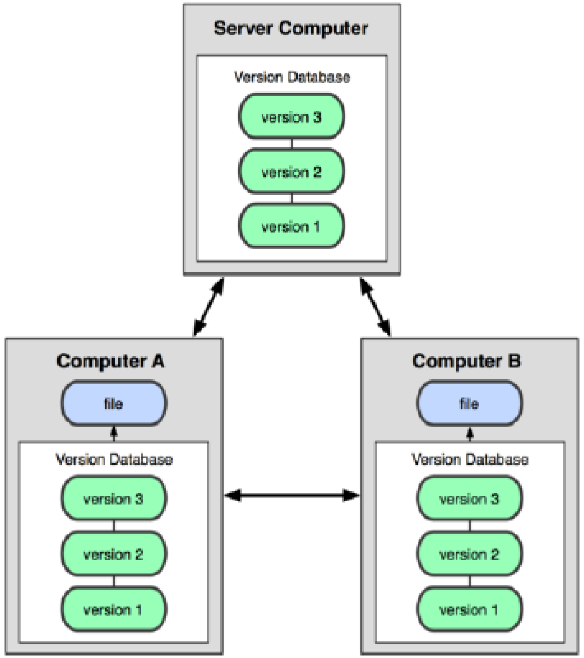
Distributed VCS (Git)

- In git, mercurial, etc., you don't "checkout“ from a central repo
  - you "clone" it and "pull" changes from it

- Your local repo is a complete copy of everything on the remote server
  - yours is "just as good" as remote server

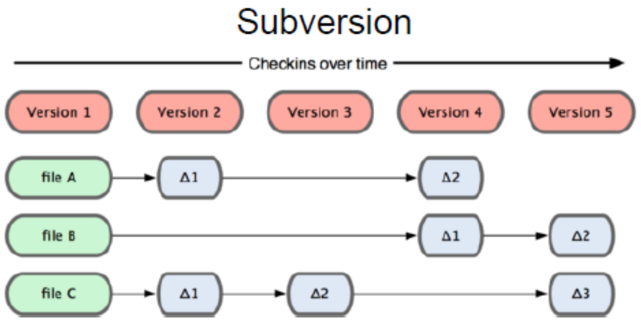
- Many operations are local:
  - check in/out from local repo
  - commit changes to local repo
  - local repo keeps version history

- When you're ready, you can "push" changes back to server

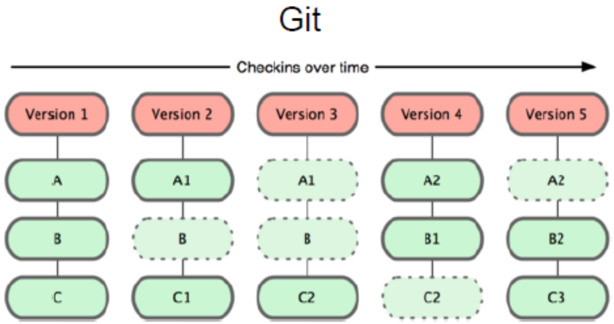


Git
Git snapshots

- Centralized VCS like Subversion track version data on each individual file.



- Git keeps "snapshots" of the entire state of the project.
  - Each checkin version of the overall code has a copy of each file in it.
  - Some files change on a given checkin, some do not.
  - More redundancy, but faster.

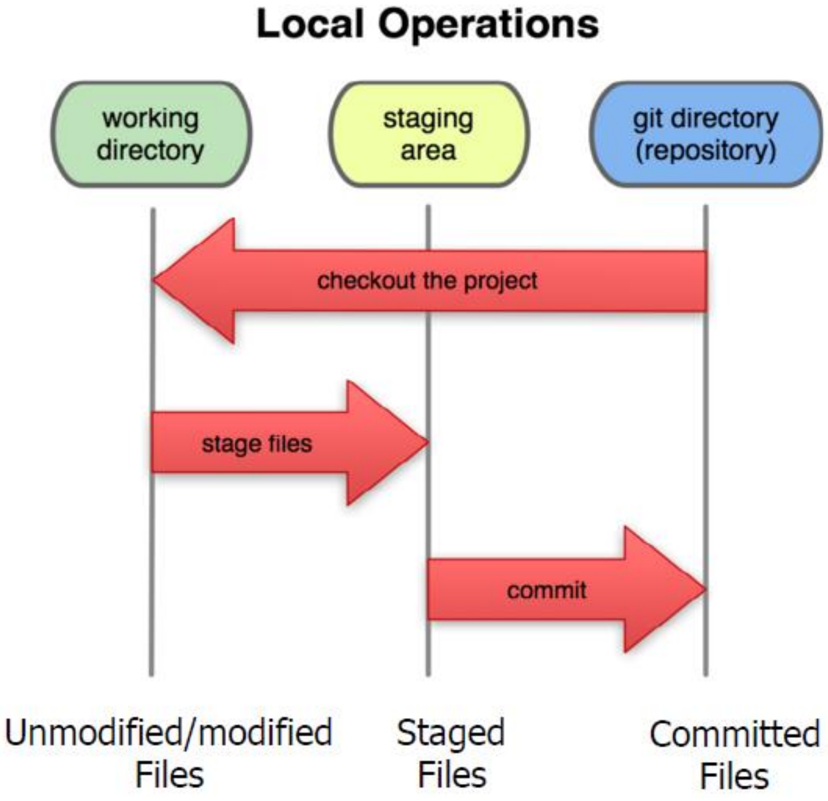


Git
Local Git areas

- In your local copy on git, files can be:
  - In your local repo
    (committed)

  - Checked out and modified, but not yet committed
    (working copy)

  - Or, in-between, in a **"staging" area**
    - Staged files are ready to be committed.
    - A commit saves a snapshot of all staged state.



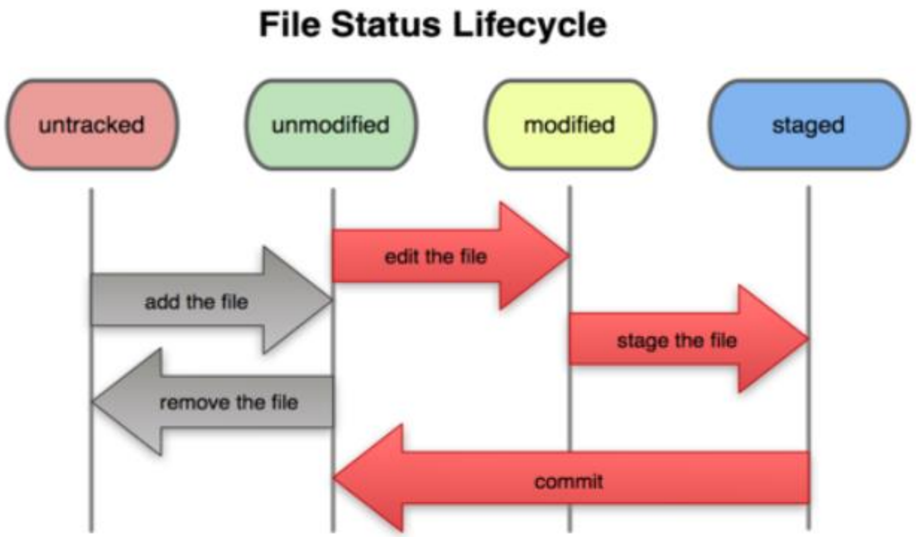
Git

Basic Git workflow

- **Modify** files in your working directory.

- **Stage** files, adding snapshots of them to your staging area.

- **Commit**, which takes the files in the staging area and Stores that snapshot permanently to your Git directory.



Git

Initial Git configuration

- Set the name and email for Git to use when you commit:

```
– git config --global user.name “Kumaraswamy"
– git config --global user.email Kumaraswamy.VenkataswamyBangalore@in.bosch.com
```

  You can call the below command to verify these are set.

```
 – git config –list
```

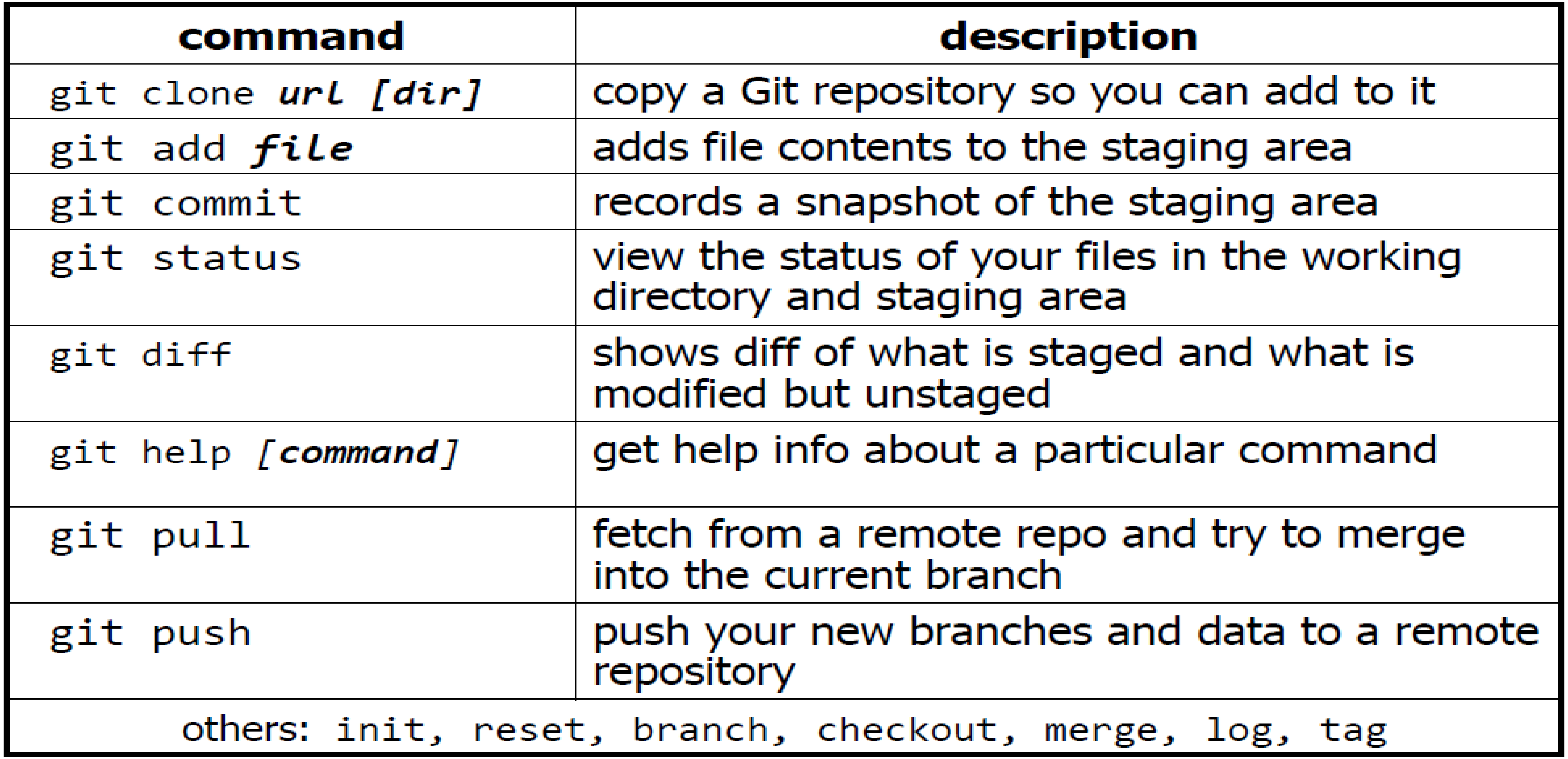
- Set the editor that is used for writing commit messages:

```
– git config --global core.editor nano
```

  (it is vim by default)

Git

Git commands



command	description
git clone ***url [dir]***	copy a Git repository so you can add to it
git add ***file***	adds file contents to the staging area
git commit	records a snapshot of the staging area
git status	view the status of your files in the working directory and staging area
git diff	shows diff of what is staged and what is modified but unstaged
git help ***[command]***	get help info about a particular command
git pull	fetch from a remote repo and try to merge into the current branch
git push	push your new branches and data to a remote repository
others: init, reset, branch, checkout, merge, log, tag	

Git
Creating a Git repo

Two common scenarios: (only do one of these)

To create a new **local Git repo** in your current directory:

```
– git init
```

- This will create a *.git* directory in your current directory.
- Then you can commit files in that directory into the repo.

```
– git add filename
– git commit -m "commit message"
```

To **clone a remote repo** to your current directory:

```
– git clone url localDirectoryName
```

- This will create the given local directory, containing a working copy of the files from the repo, and a *.git* directory (used to hold the staging area and your actual local repo)

Git

An example workflow

1. Clone the GIT repository

```
$ git clone https://KVK5KOR@sourcecode.socialcoding.bosch.com/scm/bcai/proj-ai-s-p422-ps-ec-ecu.git
```

2. To fetch all the latest checked in files:

```
$ git pull
```

3. Add the latest document:

```
$ git add PS_EC_ECU\proj-ai-s-p422-ps-ec-ecu\Data engineering\SDOM\PS_EC_GetImplementationEditor.java
```

   To add multiple files:

```
$ git add --all
```

4. Check the added file:

```
$ git status
```

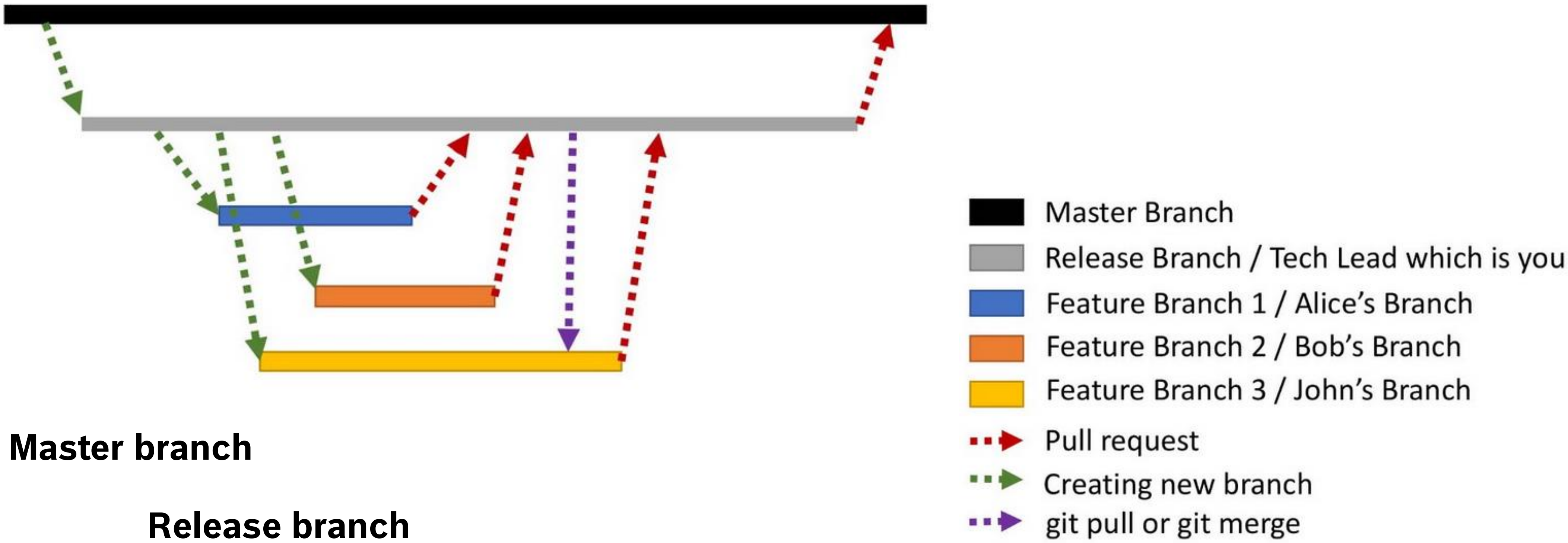
5. Commit the updated file in the local machine:

```
$ git commit -m "updated code"
```

6. Push the updated file to GIT server:

```
$ git push
```

Git
An example workflow


⬛ Master Branch
▨ Release Branch / Tech Lead which is you
🟦 Feature Branch 1 / Alice's Branch
🟧 Feature Branch 2 / Bob's Branch
🟨 Feature Branch 3 / John's Branch
▪▪▶ Pull request
▪▪▶ Creating new branch
▪▪▶ git pull or git merge

Master branch

Release branch

Feature branch

Git
Branching and merging

Git uses branching heavily to switch between multiple tasks.

- To create a new local branch:

  ```
  git branch name
  ```

- To list all local branches: (* = current branch)

  ```
  git branch
  ```

- To switch to a given local branch:

  ```
  git checkout branchname
  ```

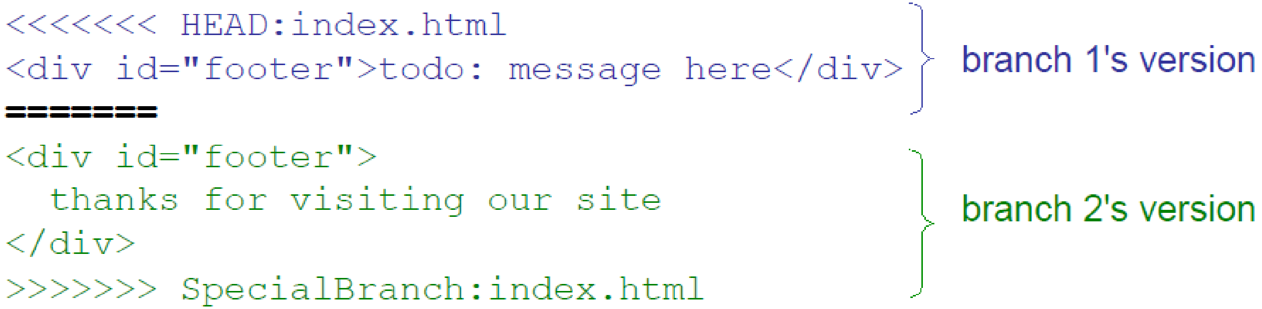
- To merge changes from a branch into the local master:

  ```
  git checkout master
  git merge branchname
  ```

Git

Merge conflicts

- The conflicting file will contain <<< and >>> sections to indicate where Git was unable to resolve a conflict:



```
<<<<<<< HEAD:index.html
<div id="footer">todo: message here</div>
=======
<div id="footer">
  thanks for visiting our site
</div>
>>>>>>> SpecialBranch:index.html
```

- Find all such sections, and edit them to the proper state (whichever of the two versions is newer / better / more correct).

Git
Viewing/undoing changes

- To view status of files in working directory and staging area:
  - `git status` or `git status -s` (short version)

- To see what is modified but unstaged:
  - `git diff`

- To see a list of staged changes:
  - `git diff -cached`

- To see a log of all changes in your local repo:
  - `git log` or `git log --oneline` (shorter version)

    ```
    1677b2d Edited first line of readme
    258efa7 Added line to readme
    0e52da7 Initial commit
    ```

  - `git log -5` (to show only the 5 most recent updates), etc.

Git

Interaction with remote repo

- **Push** your local changes to the remote repo.

- **Pull** from remote repo to get most recent changes.
  - (fix conflicts if necessary, add/commit them to your local repo)

- To fetch the most recent updates from the remote repo into your local repo, and put them into your working directory:
  - *`git pull origin master`*

- To put your changes from your local repo in the remote repo:
  - *`git push origin master`*

Git
Best Practices

1. Have *readme.md* file for every folder.
2. It is a good practice to keep your local repository code up-to-date with the code in the remote repository. This avoids a lot of merge conflicts later when you raise a pull request.
3. Using the right git workflows would ensure only one person works on one feature branch at a time. The release branch would be handled by the tech lead or a senior developer.
4. Always have a JIRA ticket to work on the features or development so its easy to track the issues of the features that was developed.
5. Master and Develop Branch only Lead’s should have permission

Git

Going Wrong with Git

1. Force Push to Remote Repository
   - whenever you are in a position where you have to use force push, use it only as a last resort. Also evaluate if there is any other way of achieving what you want without using force push.
2. Trying to Rebase the Remote repository
   - Rebasing the remote repository will alter the commit history and will create issues when other developers try to pull the latest code from the remote repository.
3. Amending commits in the remote repository
4. Hard reset (git reset --hard)

Git
References

1. Free on-line book:

   http://git-scm.com/book

2. Git tutorial:

   http://schacon.github.com/git/gittutorial.html

3. Git for Computer Scientists:

   http://eagain.net/articles/git-for-computer-scientists/

THANK YOU