Understanding Git

lynxbee.com

Version Control System - Why ?

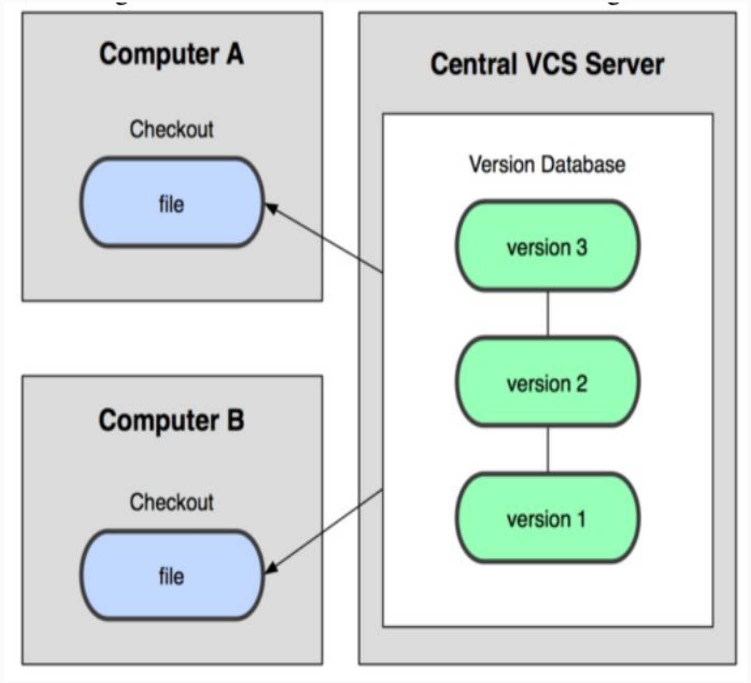
lynxbee.com

What is version control ?
=> Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

- want to keep every version
- allows you to revert files back to a previous state
- Revert the entire project back to a previous state
- compare changes over time
- See who last modified something that might be causing a problem
- who introduced an issue and when

Centralised Version Control System

lynxbee.com



- Came to existence when people need to collaborate with developers on other systems.
- have a single server that contains all the versioned files, and a number of clients that check out files from that central place.

Example - CVS, Subversion, and Perforce

Advantages - everyone knows to a certain degree what everyone else on the project is doing.

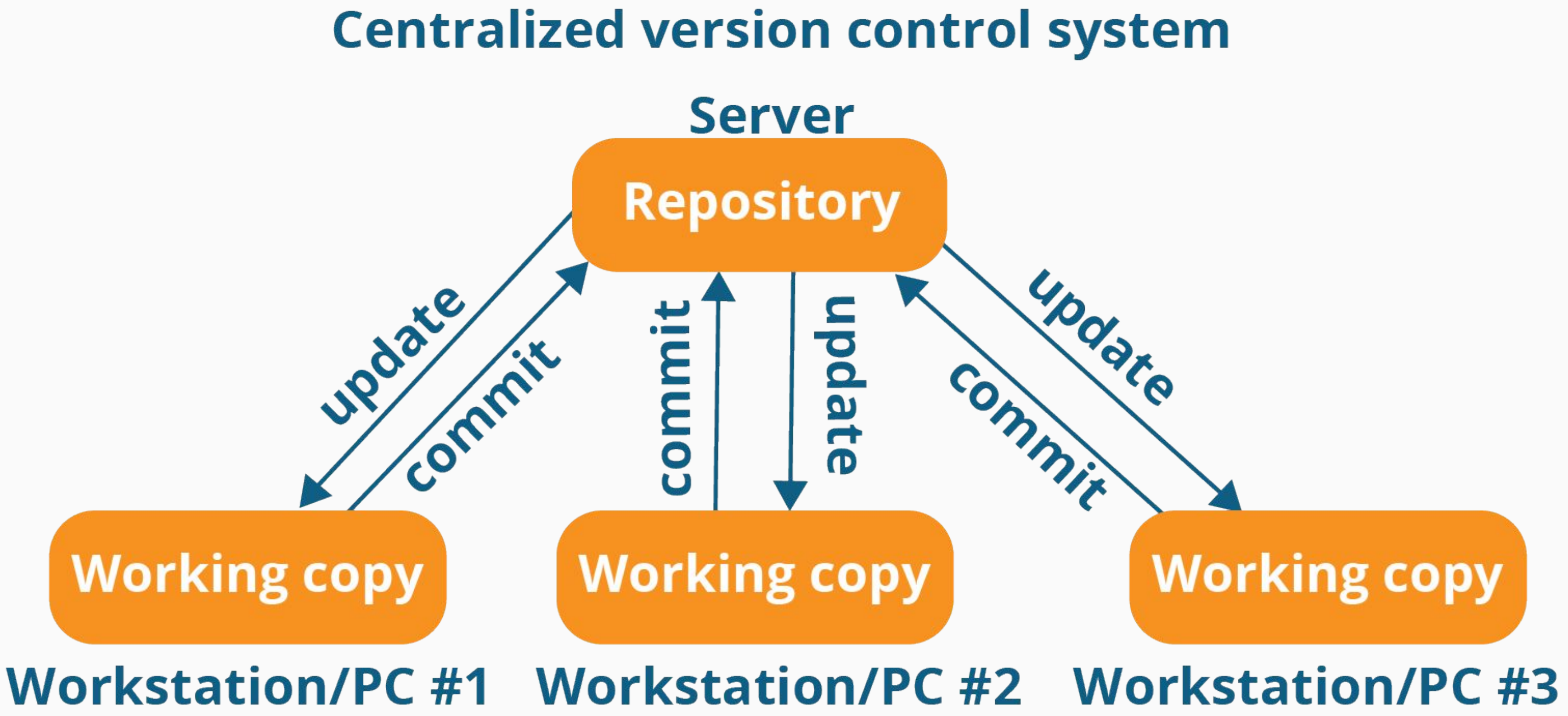
- Easier to Administer than local version control systems.

Disadvantages - single point of failure

-

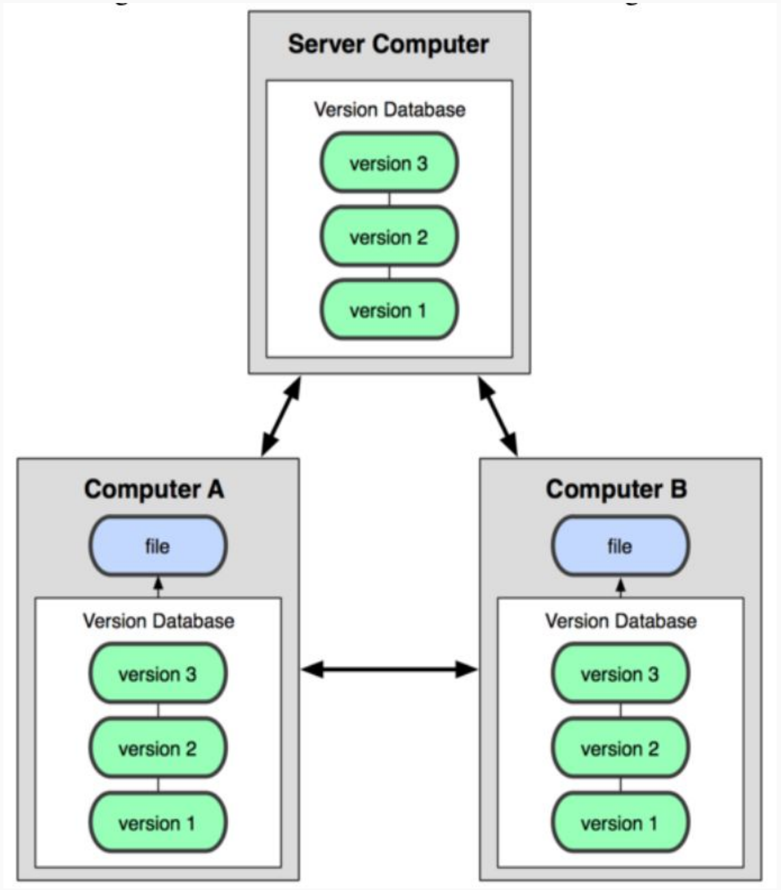
Centralised Version Control System

lynxbee.com



Distributed Version Control System

lynxbee.com



Advantages over previous version control systems

- Not a single point of failure as people can mirror entire repository / server on local PC

Example - Git, Mercurial, Bazaar or Darcs

Why Git

- Faster
- Simple to learn
- Fully Distributed
- Can handle Large projects
- Multi branch
- Stores data as snapshot
- Every operation on git it local and no dependence on remote server
- Entire history of projects can reside on local PC
- Local diff between past and present files
- No dependency on network, hence work offline and push changes to server when internet is available
- Git has integrity check with SHA-1 hash checksum hence prevents data corruption during transmission to server

Three Stage of Git

lynxbee.com

Modified, Committed and Staged

Modified - means that you have changed the file but have not committed it to your database yet.

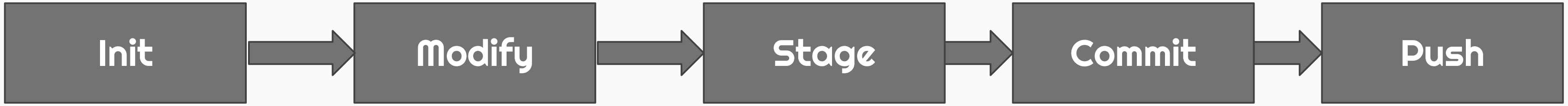
Committed - means that the data is safely stored in your local database.

Staged - means that you have marked a modified file in its current version to go into your next commit snapshot.

The staging area is a simple file, generally contained in your Git directory, that
stores information about what will go into your next commit. It's sometimes referred
to as the index, but it's becoming standard to refer to it as the staging area.

The Git Workflow

lynxbee.com



The basic Git workflow goes something like this:

1. You modify files in your working directory.

2. You stage the files, adding snapshots of them to your staging area. (simple “git add”)

3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

If a particular version of a file is in the git directory, it’s considered committed. If it’s modified but has been added to the staging area, it is staged. And if it was changed since it was checked out but has not been staged, it is modified.

Installing Git

lynxbee.com

```
$ sudo apt-get install git

$ mkdir workspace

$ git init
Initialized empty Git repository in
/home/user/workspace/code/github/git-basi
cs/.git/
```

“.git” directory is where Git stores the metadata and object database for your project.

```
$ tree .git/
.git/
├── branches
├── config
├── description
├── HEAD
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── prepare-commit-msg.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   └── update.sample
├── info
│   └── exclude
├── objects
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags
```

Identity and Editor ... One time Setup

lynxbee.com

```
$ git config --global user.name "My Name"

$ git config --global user.email “name@mycompany.com”
```

Setting Global Editor for Git

```
$ git config --global core.editor vim
```

All Global configs gets stored at

```
$ cat $HOME/.gitconfig
[user]
    email = name@mycompany.com
    name = My Name
[core]
    editor = vim
```

```
$ git config --list
user.email=name@mycompany.com
user.name=My Name
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true

$ git config --global merge.tool vimdiff
```

core.repositoryFormatVersion - Internal variable identifying the repository format and layout version.

filemode - set true means file mode permission changes are considered changes.

bare - set true means the directory is not a working directory (no real files).

Creating First Program ... Step 1 – Modify

lynxbee.com

```
$ pwd
/home/user/workspace/code/github/git-basics/

$ vim helloworld.c

#include <stdio.h>

int main(int argc, char **argv) {
        printf("Hello World\n");
        return 0;
}
```

```
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will
be committed)

    helloworld.c
```

Add a File to Git ... Step 2 - Staging

lynxbee.com

```
$ git add helloworld.c
```

Or

```
$ git add *.c
```

Or

```
$ git add .
```

Staging - “Changes to be committed”

```
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   helloworld.c
```

Commit Message to Git

lynxbee.com

```
$ git commit -m "This is Hello World Program"
[master (root-commit) f1295d3] This is Hello
World Program
 1 file changed, 6 insertions(+)
 create mode 100644 helloworld.c
```

Git log

```
$ git log
commit f1295d33d900a9b79f4e3a8272d448b1dc04947d
Author: My Name <my.name@email.com>
Date:   Thu Aug 1 05:08:05 2017 +0530

        This is Hello World Program
```

Git log ...

```
$ git log -p
commit f1295d33d900a9b79f4e3a8272d448b1dc04947d
Author: My Name <my.name@email.com>
Date:   Thu Aug 1 05:08:05 2017 +0530

        This is Hello World Program

diff --git a/helloworld.c b/helloworld.c
new file mode 100644
index 0000000..b88f634
--- /dev/null
+++ b/helloworld.c
@@ -0,0 +1,6 @@
+#include <stdio.h>
+
+int main(int argcc, char **argv) {
+       printf("Hello World\n");
+       return 0;
+}
```

Skipping from Git

lynxbee.com

```
$ touch test.txt

$ echo “This is a Test Text File” > test.txt

$ ls -alh
.git
helloworld.c
test.txt

$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    test.txt
```

Skipping from Git ...

lynxbee.com

```
$ vim .gitignore
*.txt

$ ls -alh
.git
.gitignore
helloworld.c
test.txt

$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .gitignore

nothing added to commit but untracked files present (use "git add" to track)
```

Track .gitignore

```
$ git add .gitignore

$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   .gitignore
```

To see the changed already staged into git, for the next commit. (Similar to git diff which works for modified code)

```
$ git diff --cached

$ git commit -m "Added .gitignore"
[master 6509fab] Added .gitignore
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore
```

Adding Code to Existing Source Code

lynxbee.com

Modify our helloworld.c to add two numbers and print addition.

```
$ vim helloworld.c

#include <stdio.h>

int add(int i, int j) {
        return (i+j);
}

int main(int argcc, char **argv) {
        int r;
        printf("Hello World\n");

        r = add(10,2);
        printf("Addition = %d\n", r);

        return 0;
}
```

What happens after modifying existing source code

lynxbee.com

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

	modified:   helloworld.c

no changes added to commit (use "git add" and/or "git commit -a")
```

```
$ git diff
diff --git a/helloworld.c b/helloworld.c
index b88f634..dec07c9 100644
--- a/helloworld.c
+++ b/helloworld.c
@@ -1,6 +1,15 @@
 #include <stdio.h>

+int add(int i, int j) {
+	return (i+j);
+}
+
 int main(int argc, char **argv) {
+	int r;
 	printf("Hello World\n");
+
+	r = add(10,2);
+	printf("Addition = %d\n", r);
+
 	return 0;
 }
```

How source code deletion gets tracked ?

lynxbee.com

```
$ git diff
diff --git a/helloworld.c b/helloworld.c
index b88f634..9399653 100644
--- a/helloworld.c
+++ b/helloworld.c
@@ -1,6 +1,14 @@
 #include <stdio.h>

+int add(int i, int j) {
+       return (i+j);
+}
+
 int main(int argc, char **argv) {
-       printf("Hello World\n");
+       int r;
+
+       r = add(10,2);
+       printf("Addition = %d\n", r);
+
        return 0;
 }
```

Git commit -a & -s

lynxbee.com

```
$ git commit -a

Written two number addition function

We written two number addition program,
This is demonstration of "git commit -a"

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#       modified:   helloworld.c
#

$ git commit -a
[master 3b2e856] Written two number addition function
 1 file changed, 9 insertions(+), 1 deletion(-)
```

Create a Branch and Checkout

lynxbee.com

Check Existing Branch

```
$ git branch
* master

$ git branch training-development

$ git branch
* master
  training-development

$ git checkout training-development
Switched to branch 'training-development'

$ git branch
  master
* training-development
```

With “git log” command we can see after checkout to new branch “training-development” we got a copy of what we had exactly on “master” branch.

Add Some more code on New Branch

lynxbee.com

```
$ git diff
diff --git a/helloworld.c b/helloworld.c
index 9399653..480e3a2 100644
--- a/helloworld.c
+++ b/helloworld.c
@@ -1,5 +1,9 @@
 #include <stdio.h>

+int subtract (int i, int j) {
+	return (i-j);
+}
+
 int add(int i, int j) {
 	return (i+j);
 }
@@ -10,5 +14,8 @@ int main(int argc, char **argv) {
 	r = add(10,2);
 	printf("Addition = %d\n", r);

+	r = subtract(10,2);
+	printf("Subtraction = %d\n", r);
+
 	return 0;
 }
```

On New Branch, we added “Subtraction” function as shown in git diff here.

Now, Commit this code as,

“git commit -a -s”

```
$ git commit -a -s
[training-development fc58d9d] This is subtraction code
 1 file changed, 7 insertions(+)
```

Check with “git status” “git log” and “git branch” we are all set.

Difference between Two Branches

lynxbee.com

```
$ git diff master training-development
diff --git a/helloworld.c b/helloworld.c
index 9399653..480e3a2 100644
--- a/helloworld.c
+++ b/helloworld.c
@@ -1,5 +1,9 @@
 #include <stdio.h>

+int subtract (int i, int j) {
+	return (i-j);
+}
+
 int add(int i, int j) {
	return (i+j);
 }
@@ -10,5 +14,8 @@ int main(int argc, char **argv) {
	r = add(10,2);
	printf("Addition = %d\n", r);

+	r = subtract(10,2);
+	printf("Subtraction = %d\n", r);
+
	return 0;
 }
```

Difference Between Two Commits

lynxbee.com

```
$ git diff 6509fabd7b99618be49136245345a6f19cd8482a fc58d9d9567332febf224b865d46ed88bfe4b671

diff --git a/helloworld.c b/helloworld.c
index b88f634..480e3a2 100644
--- a/helloworld.c
+++ b/helloworld.c
@@ -1,6 +1,21 @@
 #include <stdio.h>

+int subtract (int i, int j) {
+		return (i-j);
+}
+
+int add(int i, int j) {
+		return (i+j);
+}
+
 int main(int argc, char **argv) {
-		printf("Hello World\n");
+		int r;
+
+		r = add(10,2);
+		printf("Addition = %d\n", r);
+
+		r = subtract(10,2);
+		printf("Subtraction = %d\n", r);
+
		return 0;
 }
```