GIT Training

By
ADITYA PRABHAKARA

Introducing Myself

Aditya S P (sp.aditya@gmail.com)
Freelance trainer and technologist

Boring Stuff about me:

- 14+ years of experience in development and training
- Started with Java, moved to Android and now working on Big Data Technologies

Interesting Things about me:

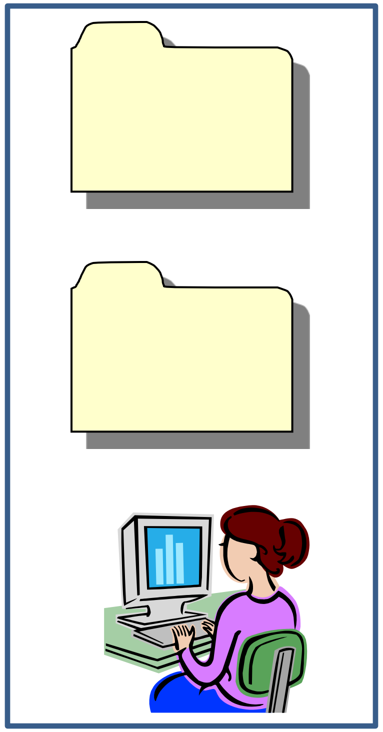
- Actually Nothing !

Getting to know you

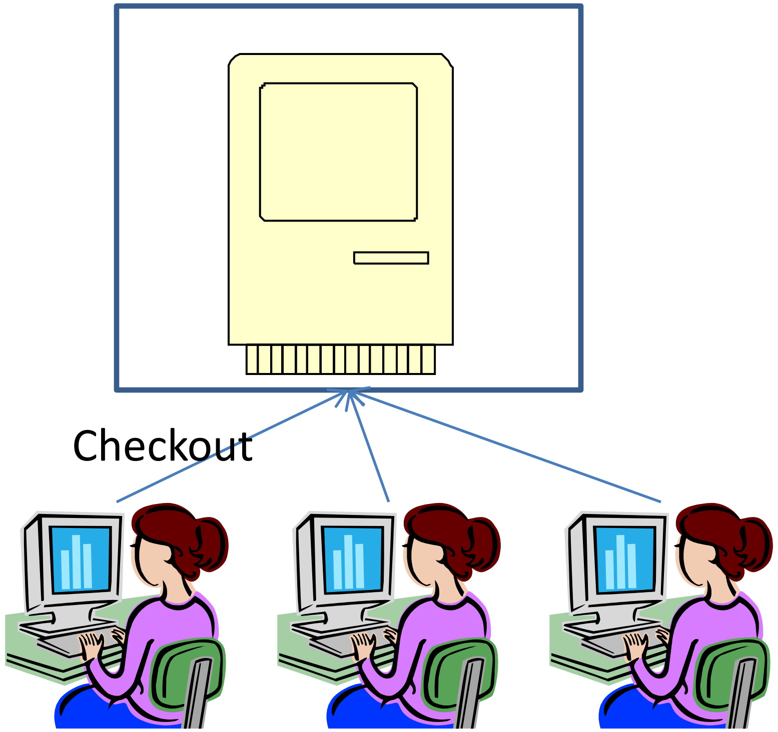
Agenda

- Git basics
- Git local
- Git remote
- Lots of hands on!

Beginnings of source control

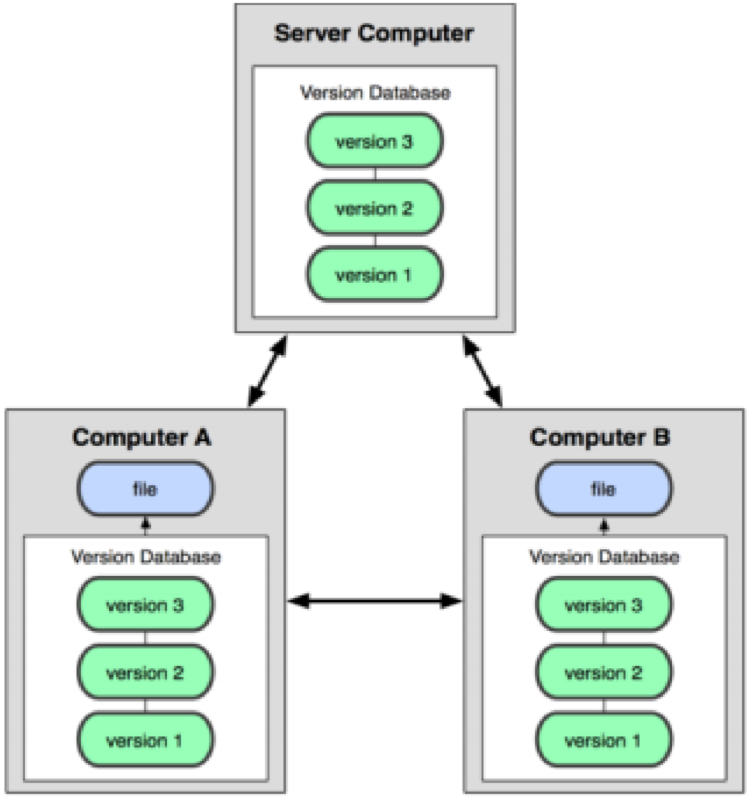


Local systems



Centralized Server

Distributed Version Control



GIT

History

Till 2002

- Linux kernel software passed around as patches and archived files

Until 2005

- Used BitKeeper as its DVCS

2005

- Linux and Linus Torvald in particular developed GIT

Base Philosophies of GIT

- Speed
- Simple design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed
- Able to handle large projects like the Linux kernel efficiently (speed and data size)

GIT Phil 1 –Snapshots not differences

- Other systems tend to store data as base versions and Delta changes to the base version

- Git stores file snapshots and links to other snapshots which have not changed.
- This makes GIT a mini file system with powerful tools built on the top of it.
- This is how GIT avoids the mistake of borrowing features from previous version control systems

GIT Phil 2 –Local Operations

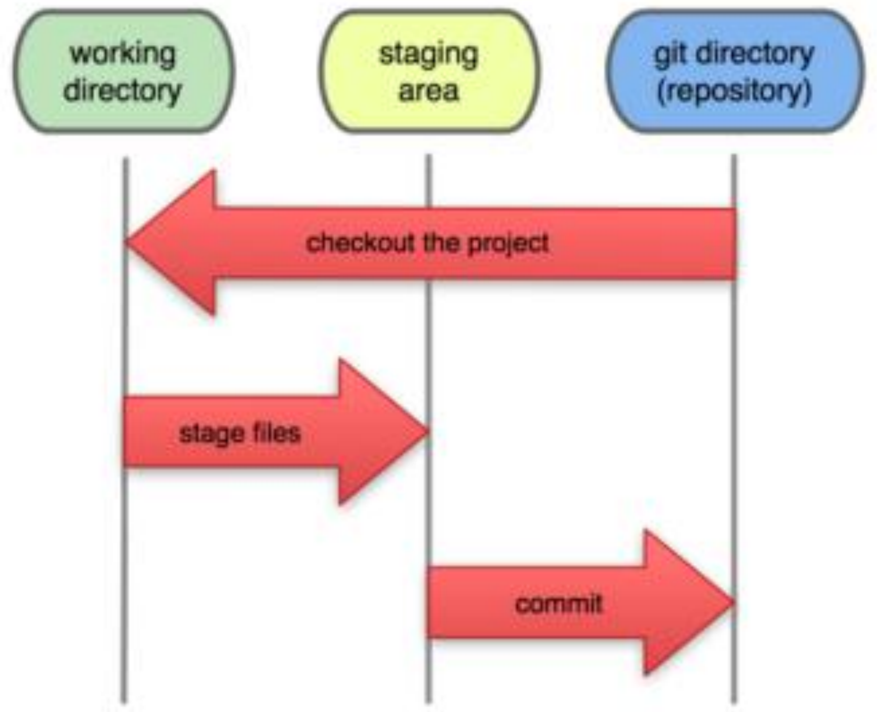
- Nearly every operation is local. No information is needed from other systems as you have the entire history stored locally

GIT Phil 3 –Integrity

- Everything is check summed
- Uses SHA-1
- You will commonly encounter strings like this

commit 62436d505c7a5099efd9de9e4afed10007866c36

GIT Phil 4 – Three states of GIT



GIT Learning – 3 stages

LOCAL	• Understand GIT commands
REMOTE	• Work with a remote GIT
TEAM	• Work with team and handle merges

GIT

GIT Setup

GIT setup

https://git-scm.com/download/win

GIT Local

GIT init

```
$ git init
Initialized empty Git repository in C:/gitexperiments/exp/.git/
```

GIT status

```
$ git status
On branch master
Initial commit
nothing to commit (create/copy files and use "git add" to track)

// create a new file readme.txt

$ git status
On branch master
Initial commit
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        readme.txt
nothing added to commit but untracked files present (use "git add" to
track)
```

GIT add (send to staging)

```
$ git add readme.txt

$ git status
On branch master

Initial commit

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

        new file:   readme.txt
```

GIT commit

```
$ git commit -m "First Commit"
[master (root-commit) 5b178d4] First Commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 readme.txt

$ git log
commit 5b178d4ace2c8bde66aa22743575b5bedb96f253
Author: AdityaSP <sp.aditya@gmail.com>
Date:   Tue Nov 22 20:34:16 2016 +0530

    First Commit
```

GIT commit

```
$ git commit -m "First Commit"
[master (root-commit) 5b178d4] First Commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 readme.txt

$ git log
commit 5b178d4ace2c8bde66aa22743575b5bedb96f253
Author: AdityaSP <sp.aditya@gmail.com>
Date:   Tue Nov 22 20:34:16 2016 +0530

    First Commit
```

GIT – Modify a tracked file

```
$ 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:   readme.txt

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

GIT stage and modify

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

        modified:   readme.txt

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:   readme.txt
```

GIT: unstage

```
$ git reset HEAD readme.txt
Unstaged changes after reset:
M        readme.txt
```

GIT: diff

```
$ git diff --staged
diff --git a/readme.txt b/readme.txt
index e69de29..934b58f 100644
--- a/readme.txt
+++ b/readme.txt
@@ -0,0 +1,2 @@
+Hi There
+something new

$ git diff 5b178d4ace2c8bde66aa22743575b5bedb96f253
c9f1d256ea21a5a898ddb73e269b6ded3118c960
diff --git a/newfile2.txt b/newfile2.txt
new file mode 100644
index 0000000..e69de29
```

"git diff –staged" will only show changes to files in the "staged" area.
"git diff HEAD" will show all changes to tracked files. If you have all changes staged for commit, then both commands will output the same.

GIT: remove files

```
$ rm newfile2.txt
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working
directory)

        deleted:    newfile2.txt

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

$ git rm newfile2.txt
rm 'newfile2.txt'
```

GIT: undo

```
git commit –amend

Will help you add some staged content back to main
$ git commit -m ’initial commit’
$ git add forgotten_file
$ git commit --amend
```

GIT: tag

```
git tag -a v1.1 -m “First tag”
git tag v1.2
git show v1.1
git show v1.2

git  tag -a v1.3 <checksum of commit>
```

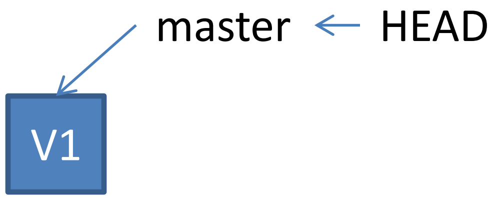
GIT: .gitignore

```
# a comment this is ignored

*.a                 # no .a files
!lib.a              # but do track lib.a, even though you're ignoring .a files above
/TODO               # only ignore the root TODO file, not subdir/TODO
build/              # ignore all files in the build/ directory
doc/*.txt           # ignore doc/notes.txt, but not doc/server/arch.txt
```

GIT: branch

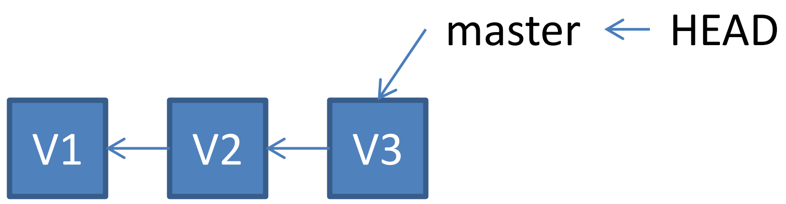
```
$ git branch
* master
```



```
So master is a branch too!
```

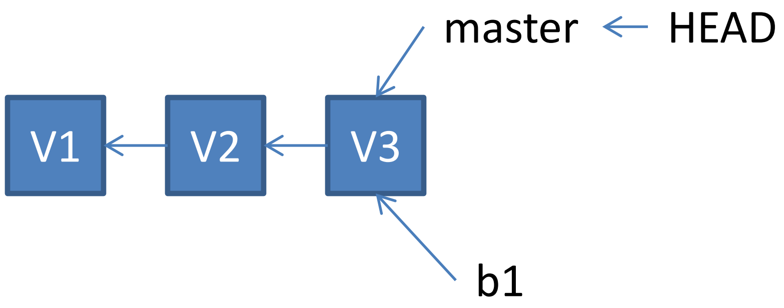
GIT: branch

```
After two commits
```



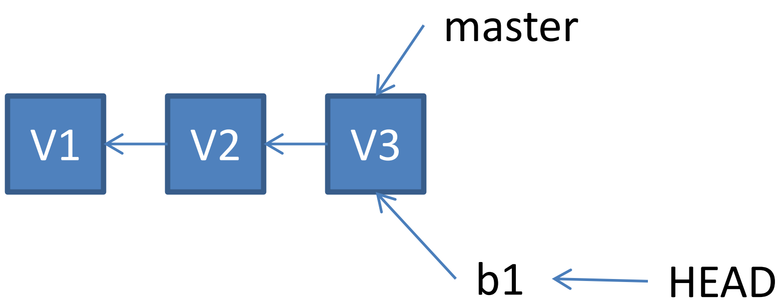
GIT: branch

```
$ git branch b1
$ git branch
  b1
* master
```



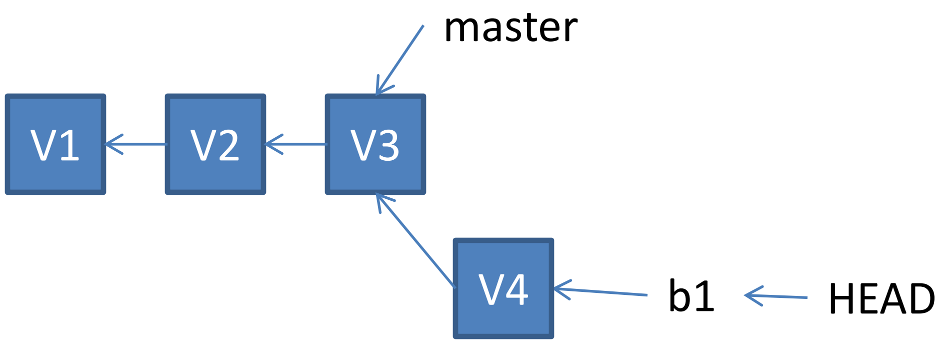
GIT: branch

```
$ git checkout b1
Switched to branch 'b1`
// Note the change in command prompt too!
```



GIT: branch

```
$ git commit -a -m "Changed in branch"
[b1 daf934f] Changed in branch
 1 file changed, 2 insertions(+)
```

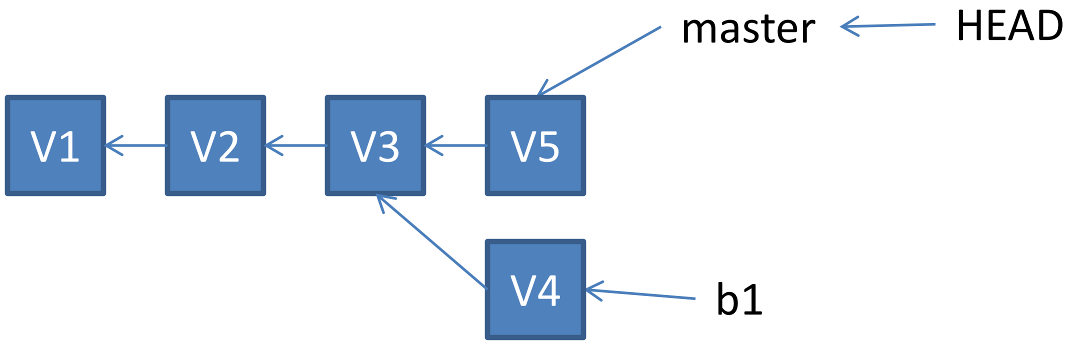


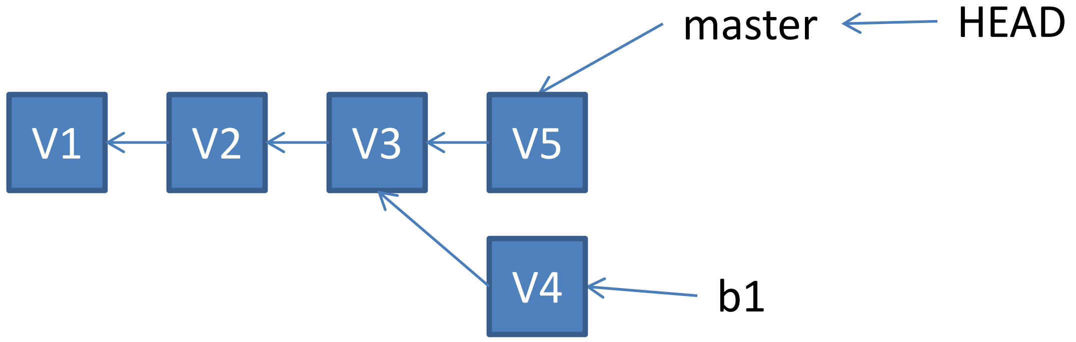
GIT: merge

```
$ git checkout master
Switched to branch 'master'
$ git merge b1
Updating c9f1d25..daf934f
Fast-forward
 readme.txt | 2 ++
 1 file changed, 2 insertions(+)
```

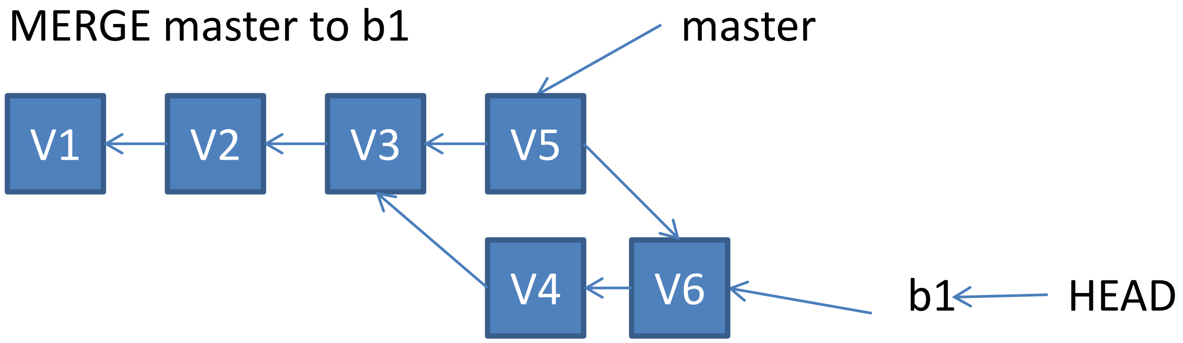
GIT: branch

```
How do we arrive at this?
```

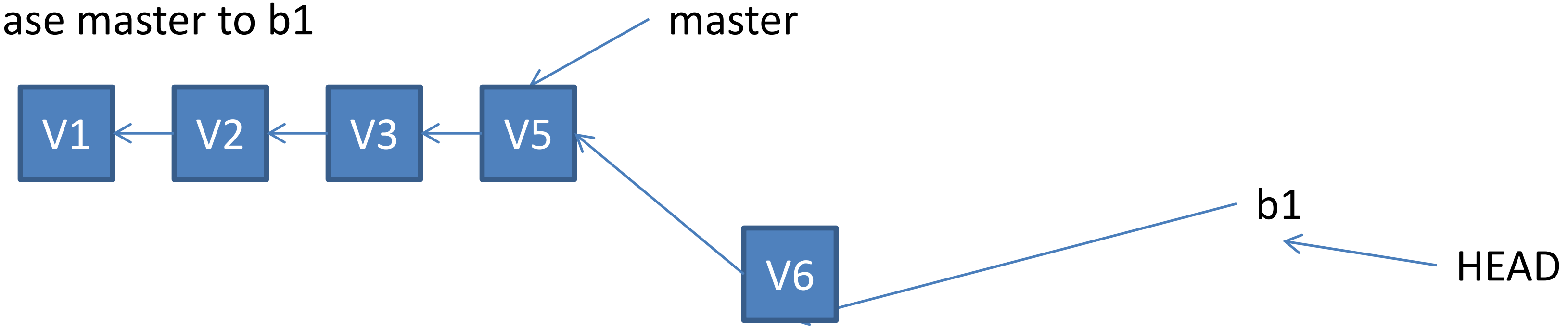




MERGE master to b1



Rebase master to b1



GIT: merge conflicts

```
Workout different scenarios
```

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

This means that, for example, instead of typing git commit, you just
need to type git ci
```

GIT

GIT Remote

GIT: start with a clone

```
$ git clone <URL>

$ git clone <URL> <DIR>
```

GIT: git remote

```
cd to the directory of your remote clone

$git remote
$git remote -v

$git remote show origin
```

GIT: fetch and pull

```
$git fetch origin
```

GIT: push

```
$git push

git push origin -tags

git push origin v1.5
```

GIT: scenario

```
1. Github with two branches
2. Clone
3. Checkout branch1
4. Make changes
5. Commit
6. Checkout master
7. Merge branch1
8. Push master
9. Checkout branch1
10.Push branch1

11.What is the difference between 8 and 10
```

GIT: tracking branch

```
1. Difference between a local branch and a tracking branch
```

GIT: branching workflows

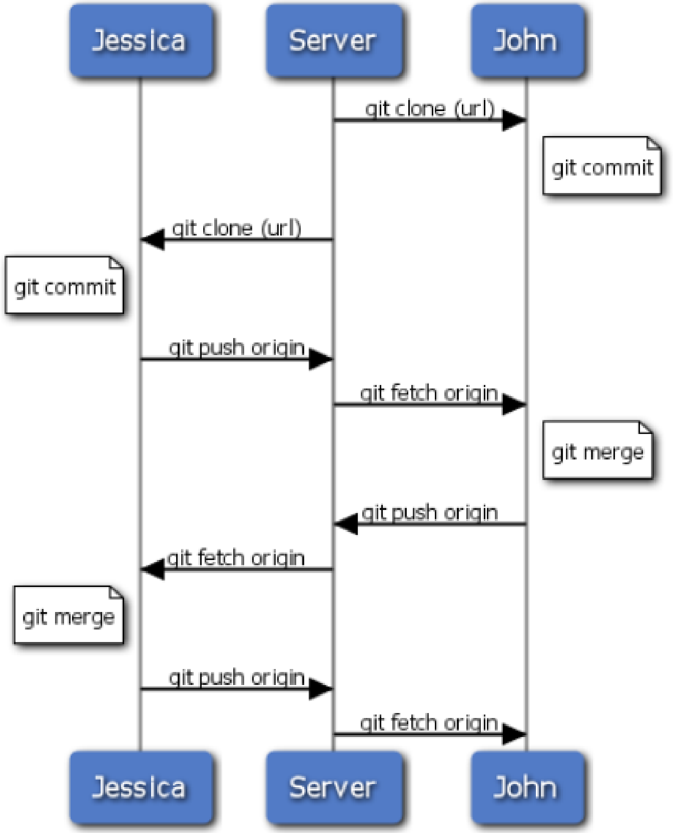
Long-Running Branches

Topic Branches

GIT: scenario tracking and local branches

```
1. Create a local branch
2. Try git push
```

GIT: working with Team


Jessica
Server
John
git clone (url)
git commit
git clone (url)
git commit
git push origin
git fetch origin
git merge
git push origin
git fetch origin
git merge
git push origin
git fetch origin
Jessica
Server
John