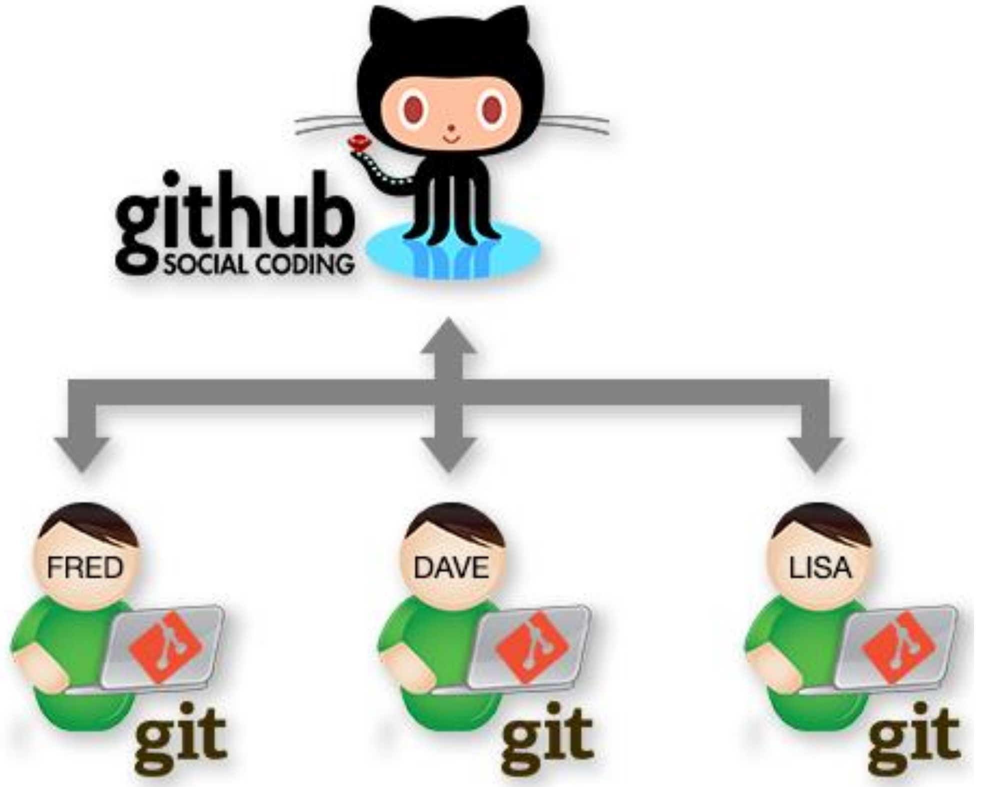
git

Intro

- Developers collaborate
  - Source control systems help teams collaborate
- Git has had a major impact on how this is achieved.
- Agenda:
  - Introduction to Git basics.
  - Collaborating with git
  - Branching

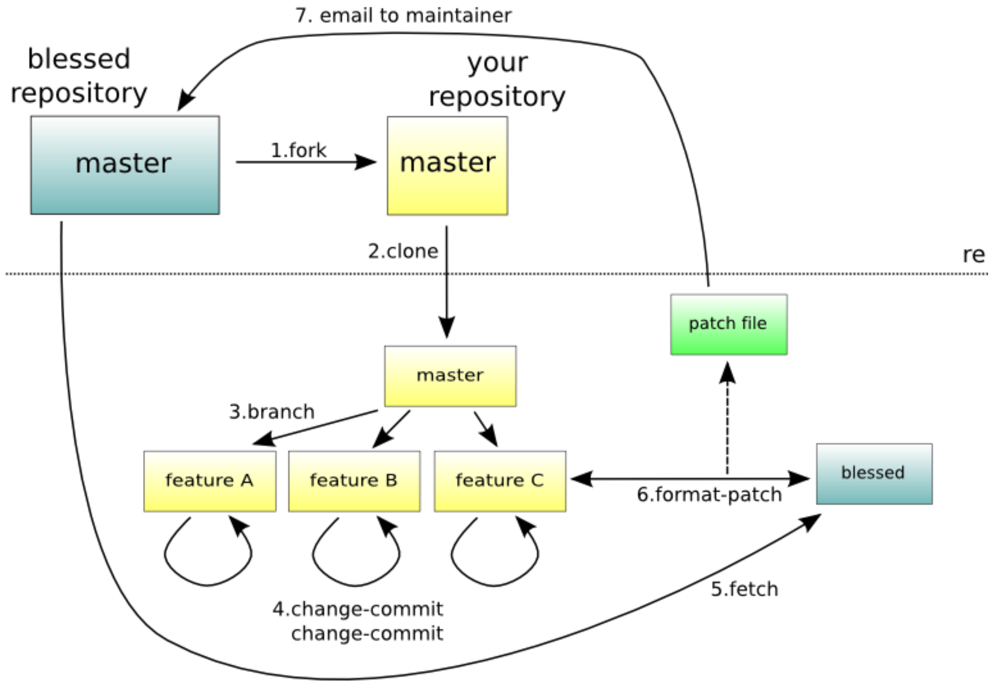


Some History

- Created by Linus Torvalds for work on the Linux kernel ~2005
- Used by:
    - Nearly everybody at this stage…

What is Git

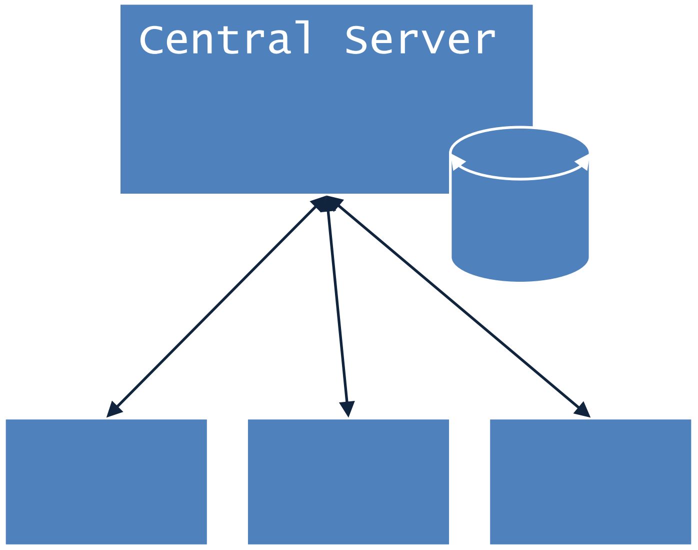
- Distributed Version Control
- Directory Content Management
- Tree Based History
- Everybody has complete history
- **I get confused sometimes…**

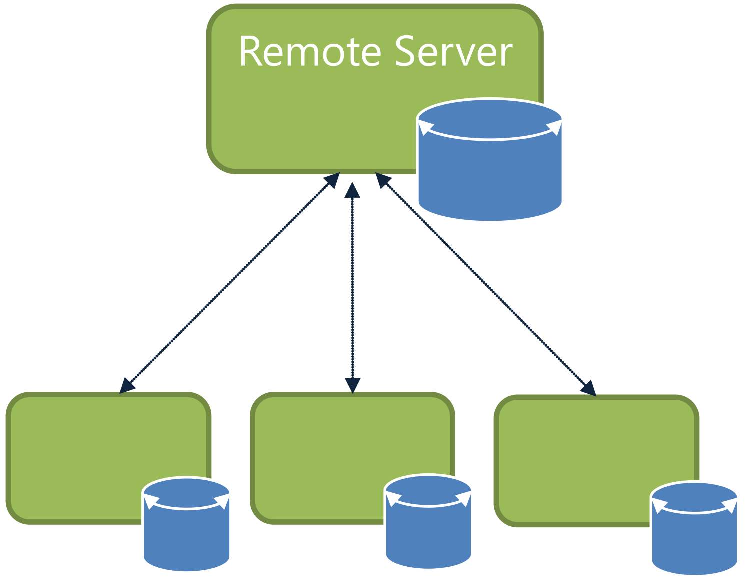


Distributed Content

- Everyone has their own copy
- Work Offline
- No Central Authority
  - Except by mutual agreement
- Changes can be shared without a server…
  - Can be configured to work peer to peer
  - Can keep collaborating even if server is gone…

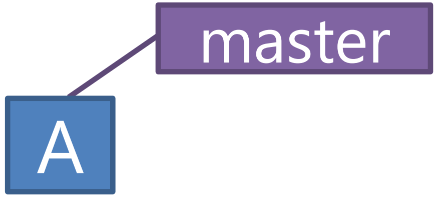
Centralised vs Distributed



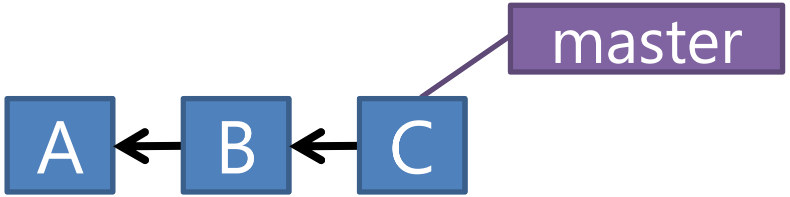


Initialising a repo…

```
$ mkdir myproject
$ cd myproject
$ git init
Initialized empty Git repository in /home/ec2-user/myproject/.git/
$ git config --global user.name “fxwalsh“
$ git config --global user.email fxwalsh@wit.com
$ vi README.txt
$ git add .
$ git commit -m 'initial commit'
[master (root-commit) 7d738f4] initial commit
1 file changed, 1 insertion(+)
 create mode 100644 README.txt
```



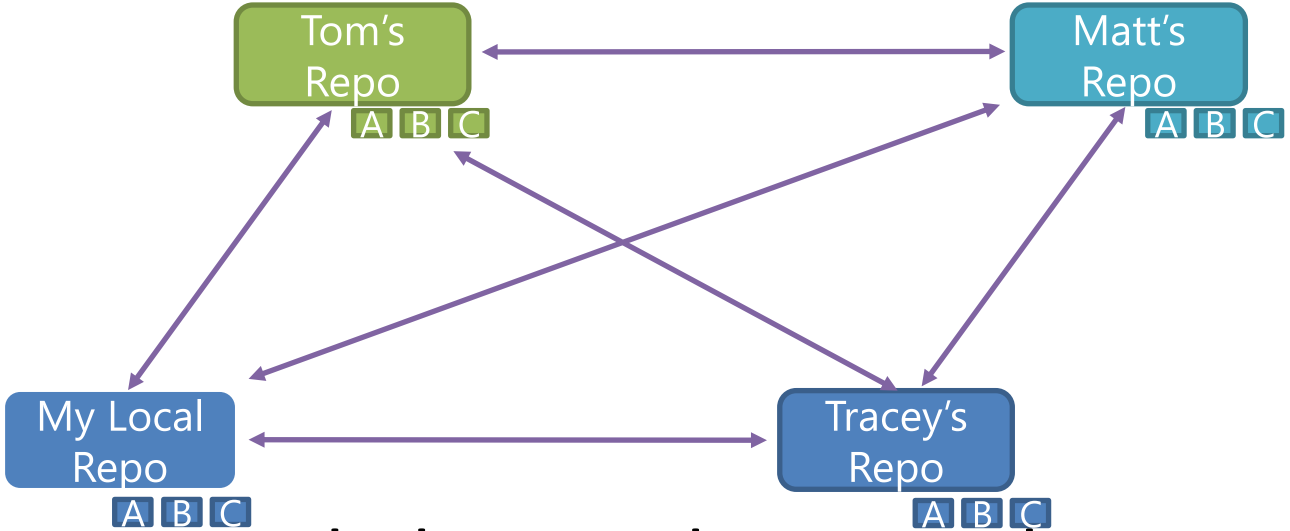
Multiple Commits



```
git commit –m “updated text file”
git commit –m “updated text file again”
```

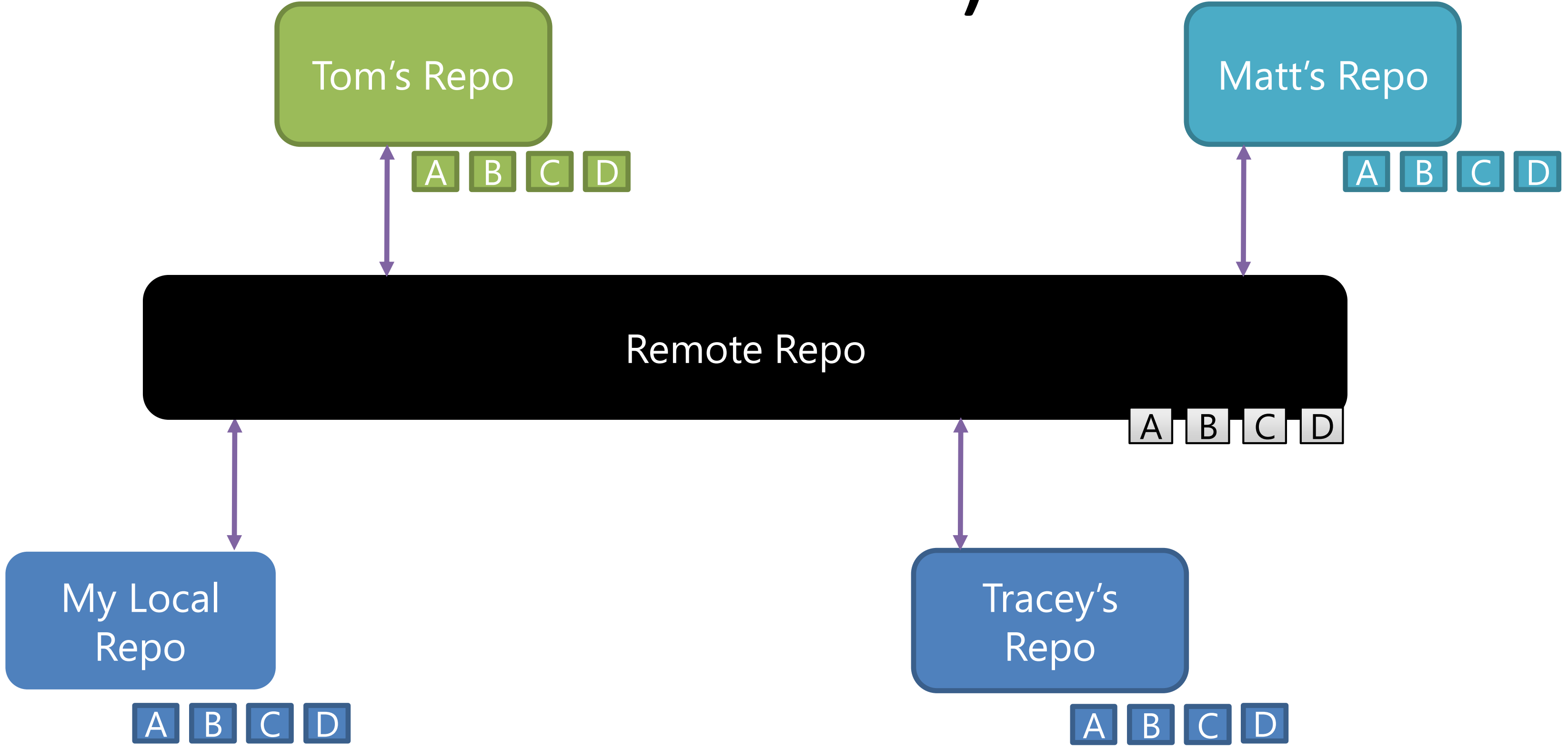
Collaborating with Git

Peer – to – Peer



•You can work this way but it might get complicated…

Central Repository (e.g. GitHub, Bitbucket)



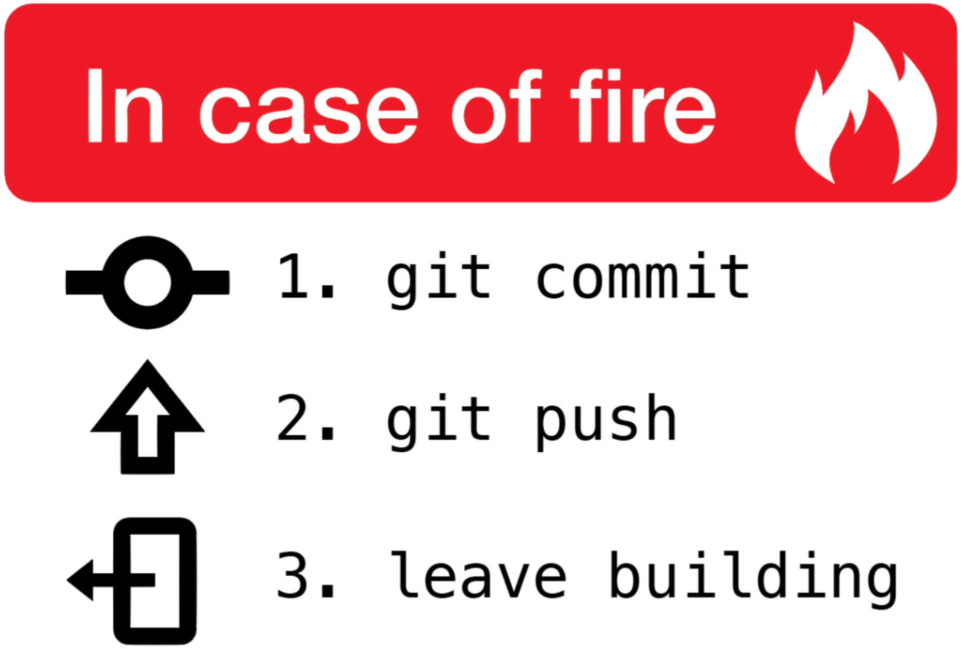
Adding a Remote Repo to Existing Project

```
 git remote add origin https://github.com/fxwalsh/BSc4Repo.git
 git remote -v
origin  https://github.com/fxwalsh/BSc4Repo.git (fetch)
origin  https://github.com/fxwalsh/BSc4Repo.git (push)
```

Setting up Remote via Cloning

```
 git clone
……
 git remote -v
origin  https://github.com/fxwalsh/BSc4Repo.git (fetch)
origin  https://github.com/fxwalsh/BSc4Repo.git (push)
```

git push



- Pushes your changes to remote
- Changes will be rejected if newer changes exist on remote
- Good to pull then push
  - merge locally, then push the results.

Pushing to remote

- By default, remote repo is labelled origin.
- Say you want to stage, commit, and push all your latest changes to origin…

```
git add --all
git commit –m “some important update to important stuff”
git push origin master
```

Potential situation 1

I can’t push to origin, it’s saying “! [rejected] master -> master (fetch first)”

- Probably changes on the remote repo that you don’t have locally.
- Solution: Do a **pull** then **push.**
- Should work as long and there’s no conflict

```
$ git push
    To https://github.com/fxwalsh/project2.git
    ! [rejected]        master -> master (fetch
    first) error: failed to push some refs
….
$git pull
…..
    Merge made by the 'recursive' strategy.
    README.md | 2 +-
    1 file changed,1 insertion(+), 1 deletion(-)
$git push
Counting objects: 5, done.
…
```

Potential situation 2

I want to replace local repo with remote, don’t care about losing changes

- Solution: fetch from the default remote, origin
- Reset your current branch (master) to origin's master

```
$ git fetch
…
$ git reset --hard origin/master
```

Potential situation 3

I want to replace local repo with remote, don’t care about losing changes on remote

- Solution: force the push

```
$ git push -f origin master
```

Potential situation 4

Trying to pull but getting "CONFLICT (content): Merge conflict in …."

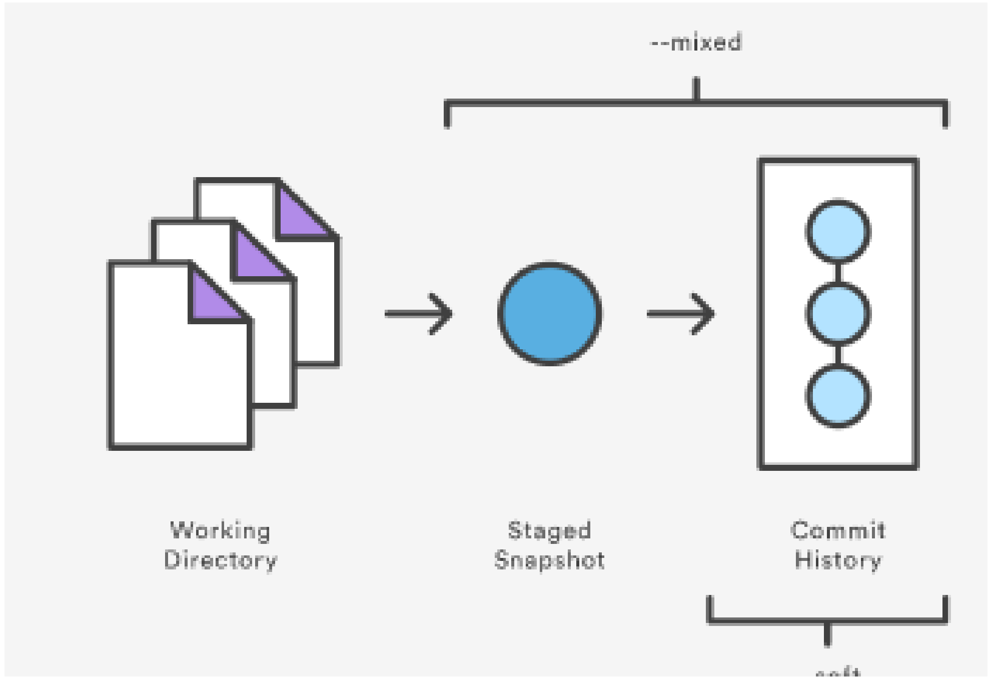
- Two different versions of the same file(s)
- Decide which one you want to keep manually.
- Perhaps force push or hard reset.

```
$ git push -f origin master
```

Potential Situation 5

I committed stuff I really didn't want to. I want to revert to the previous commit.

- Do **git reset HEAD~1**
  - **Moves Head (pointer to latest commit) back one**
  - **Leaves files alone (but not staged)**
- Do **git reset –hard HEAD~1**
  - **Gets rid of changes completely**
- Do **git reset –soft HEAD~1**
  - **Leaves changes staged**

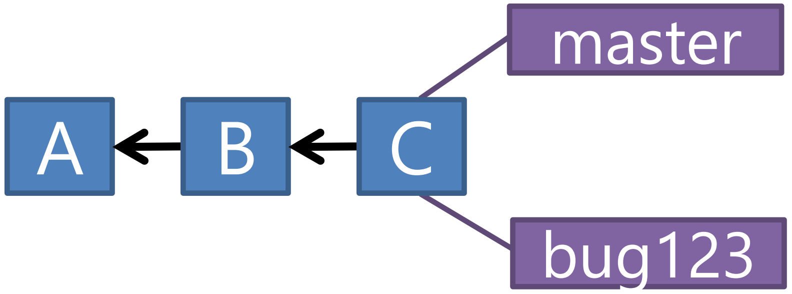


Branching with Git

Branching

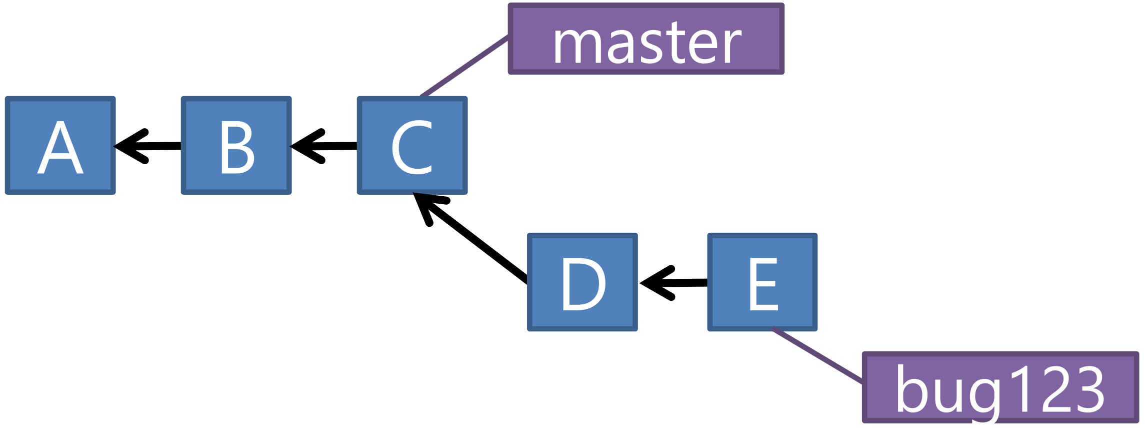
- Like a label on a graph node
- All branching takes place in the same folder/directory
  - Things might appear to disappear depending on what branch you work on…
- You can switch branches
  - Analogous to moving label from one node to another

Branching



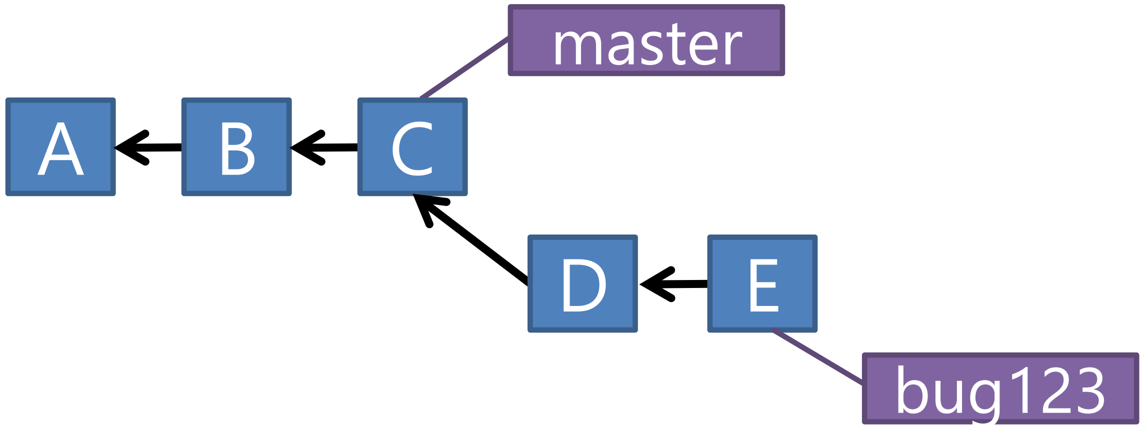
```
git checkout -b bug123
Switched to a new branch 'bug123'
```

Branching



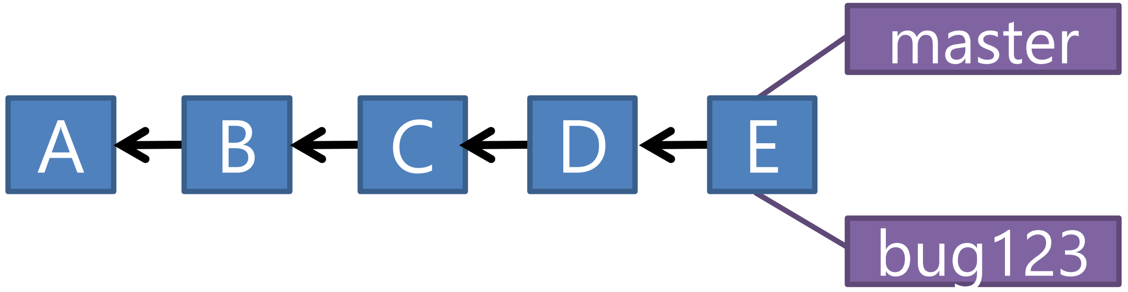
```
git commit –m “bug fix”
git commit –m “another code fix”
```

Branching



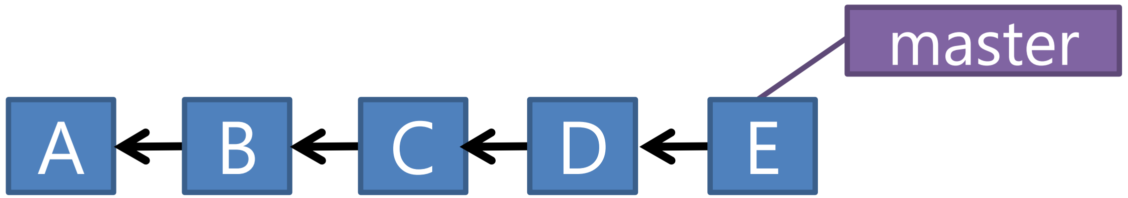
```
git checkout master
vi README.txt
```

Branching



```
git merge bug123
```

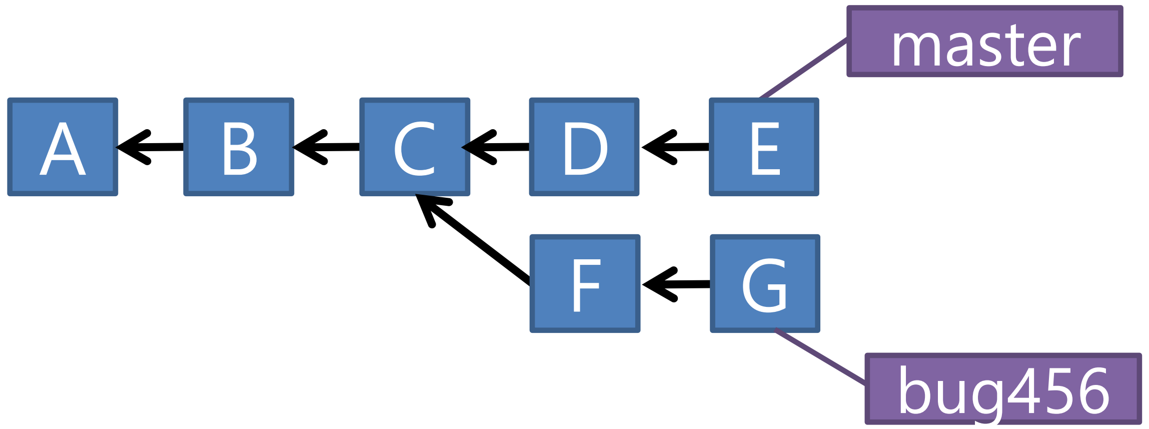
Delete Branch



```
git branch -d bug123
```

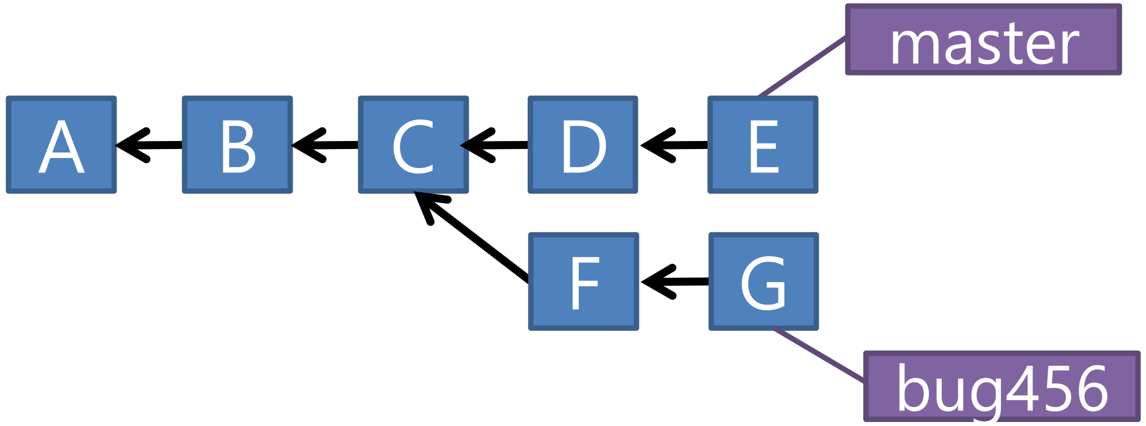
Deleted branch bug123 (was 0e85eb8).

Branching



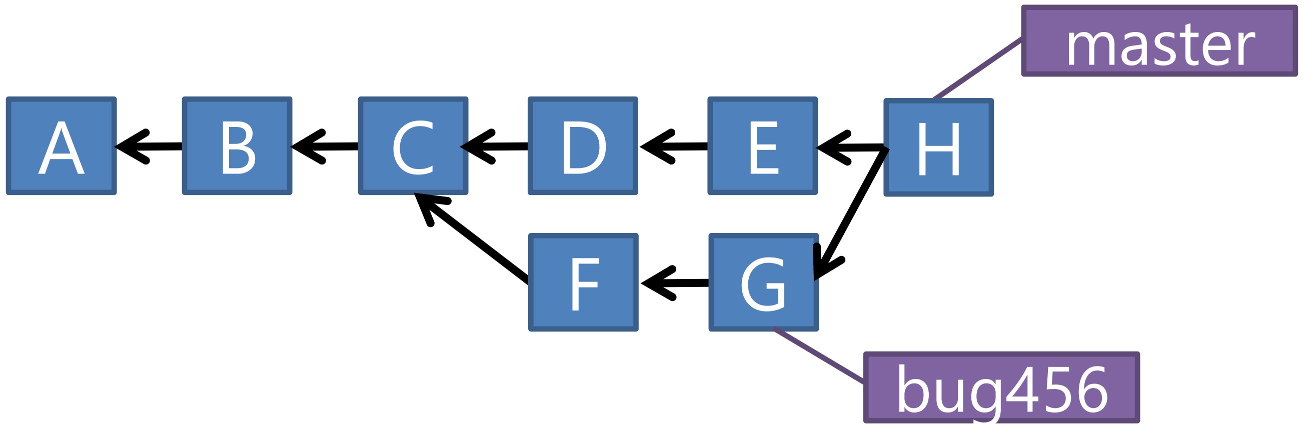
•Suppose another bug branch off of (C).
•Also, changes have happened in master (bug 123 which we just merged) since then.
•Also, two commits in bug456.

Merging



```
git checkout master
```

Merging

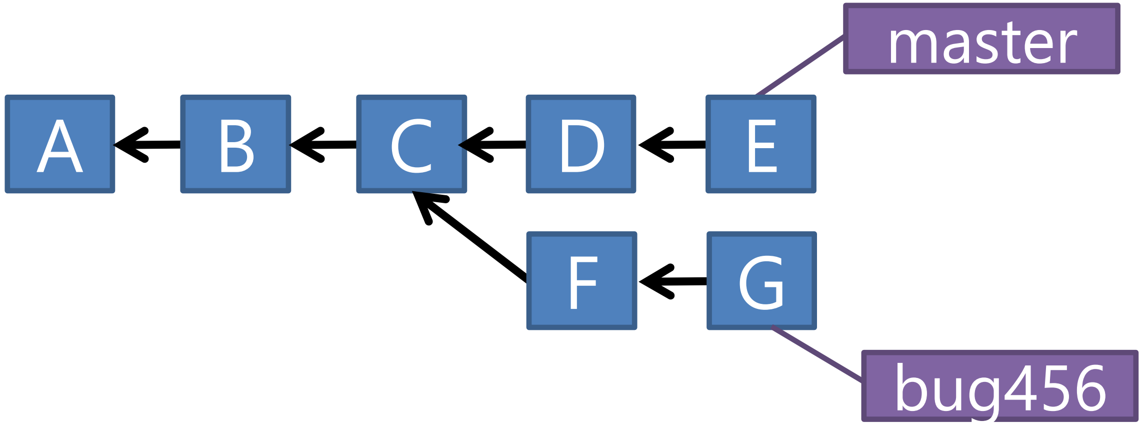


git merge bug456

if there are conflicts, they need to be resolved manually

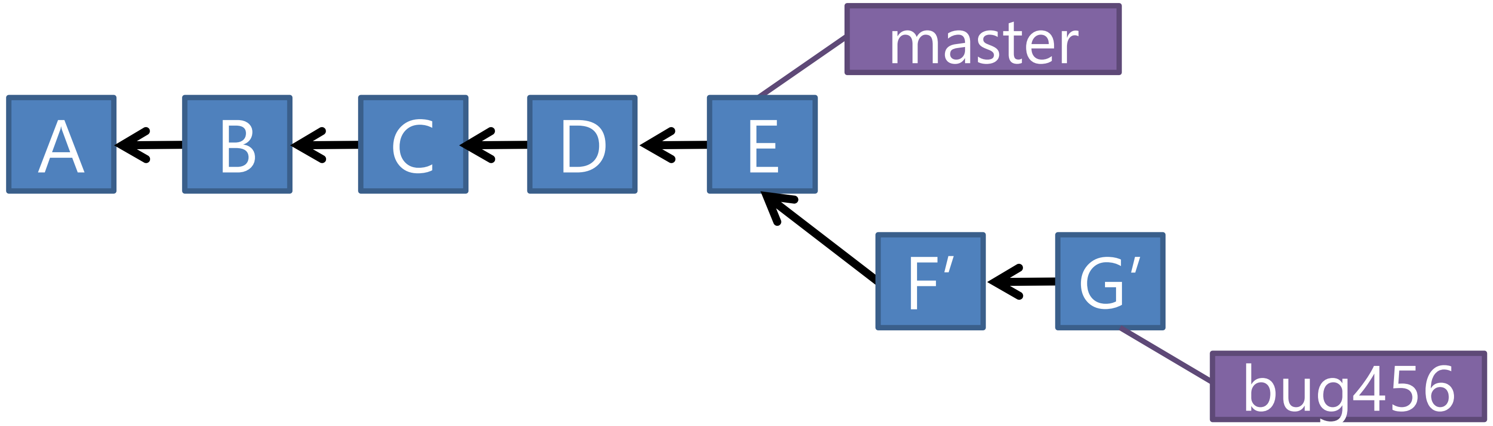
Also deleting the bug456 branch can leave a non-linear, messy structure.

Merging - Rebase



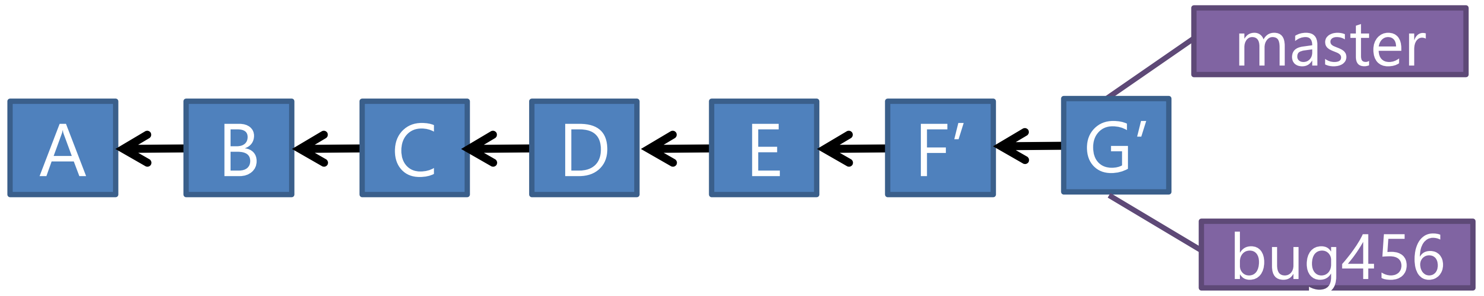
As before, but this time we rebase first….

Merging: Rebase



•Changes on (C) are undone and applied to (E) instead.

Merging: Rebase



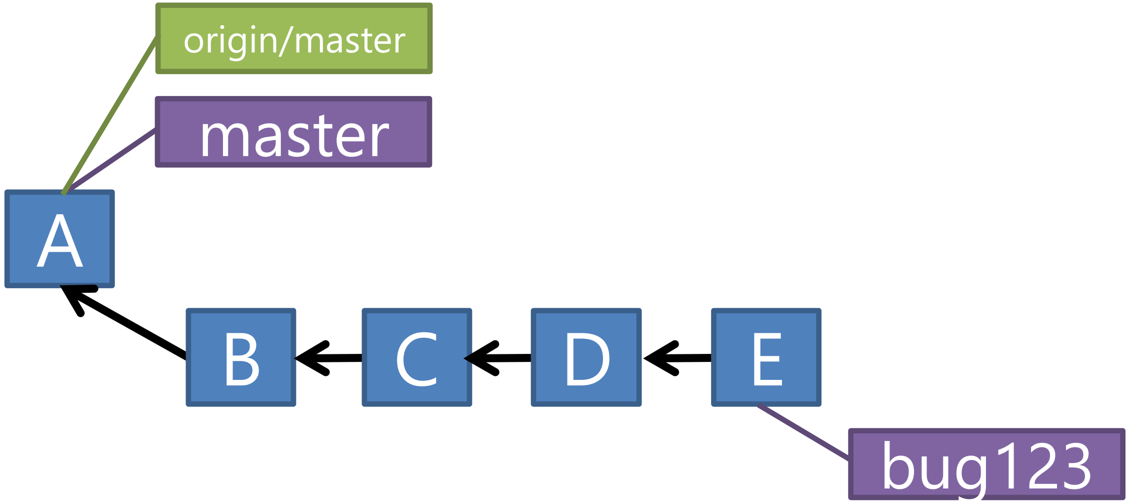
```
git checkout master
git merge bug456
```

- Linear, causal flow of changes.
- Less snapshots in repository

Branching and Merging: Key points

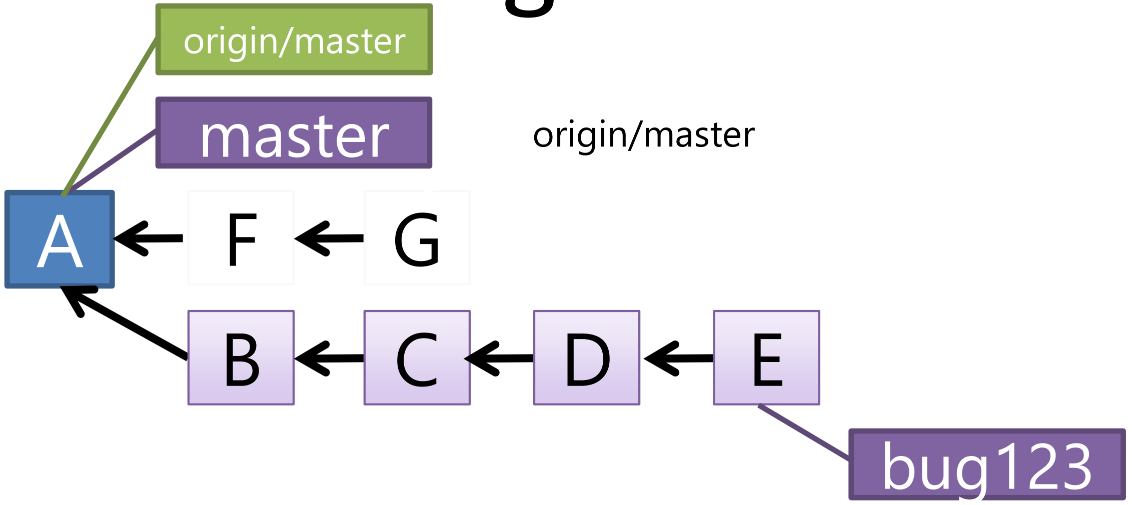
- Quick and Easy to create ‘Feature’ Branches
- Very capable tool to manage changes
- Rebasing helps keep things clearer.

Branching with Remote



Changes on Bug123 branch are only local.

Branching with Remote



- Can have situation where there’s two versions of the origin/master
  1. what was last known about the upstream master
  2. what is actually up there (which we don’t know about).

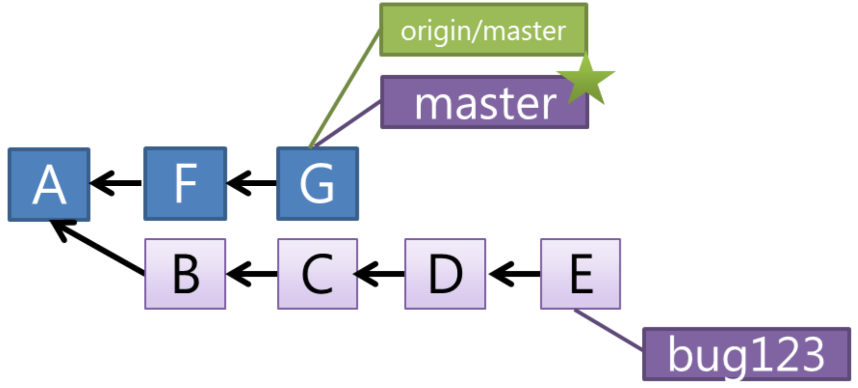
Branching with Remote

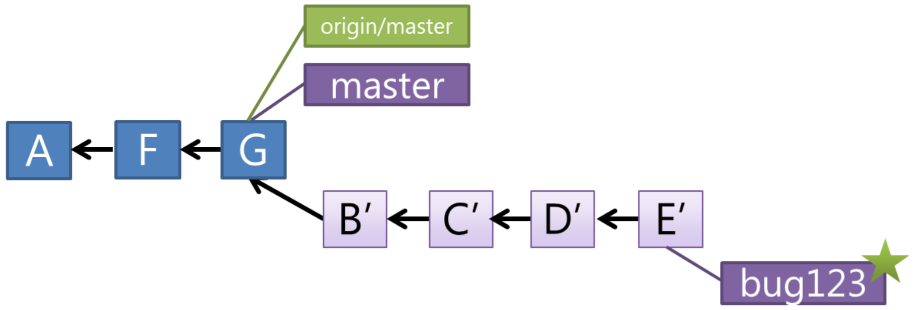
- Update Master to what's on remote

```
git checkout master
git pull origin
```

- Rebase the bug123 branch

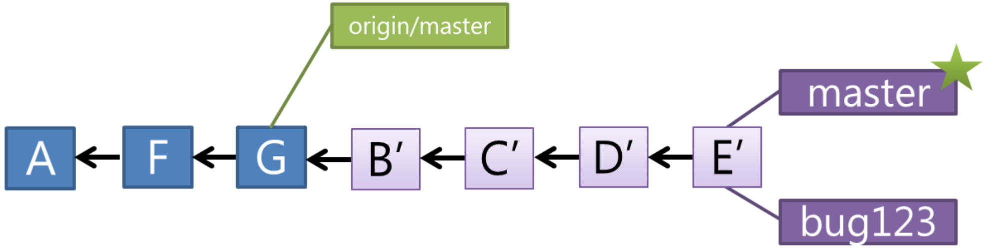
```
git checkout bug123
git rebase
```



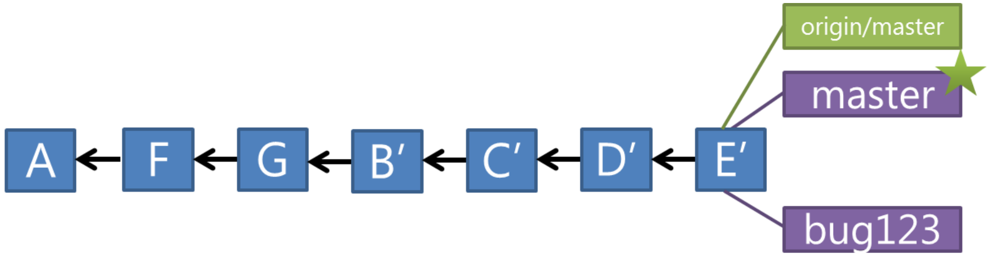


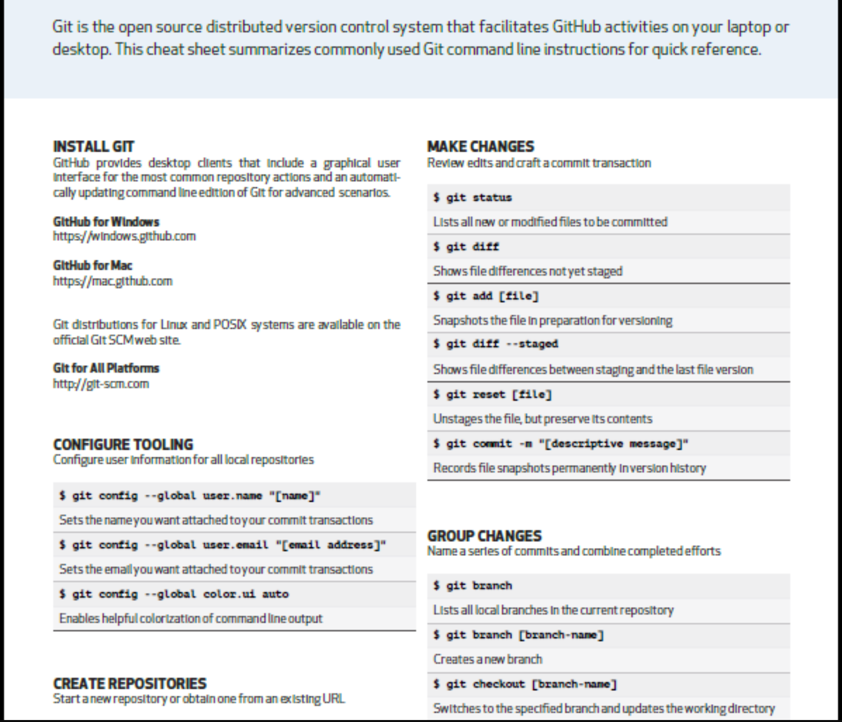
Branching with Remote

```
git merge bug123
```



```
git push origin
```





Git is the open source distributed version control system that facilitates GitHub activities on your laptop or desktop. This cheat sheet summarizes commonly used Git command line instructions for quick reference.

INSTALL GIT

GitHub provides desktop clients that include a graphical user interface for the most common repository actions and an automatically updating command line edition of Git for advanced scenarios.

GitHub for Windows
https://windows.github.com

GitHub for Mac
https://mac.github.com

Git distributions for Linux and POSIX systems are available on the official Git SCM web site.

Git for All Platforms
http://git-scm.com

CONFIGURE TOOLING

Configure user information for all local repositories

```
$ git config --global user.name "[name]"
```

Sets the name you want attached to your commit transactions

```
$ git config --global user.email "[email address]"
```

Sets the email you want attached to your commit transactions

```
$ git config --global color.ui auto
```

Enables helpful colorization of command line output

CREATE REPOSITORIES

Start a new repository or obtain one from an existing URL

MAKE CHANGES

Review edits and craft a commit transaction

```
$ git status
```

Lists all new or modified files to be committed

```
$ git diff
```

Shows file differences not yet staged

```
$ git add [file]
```

Snapshots the file in preparation for versioning

```
$ git diff --staged
```

Shows file differences between staging and the last file version

```
$ git reset [file]
```

Unstages the file, but preserve its contents

```
$ git commit -m "[descriptive message]"
```

Records file snapshots permanently in version history

GROUP CHANGES

Name a series of commits and combine completed efforts

```
$ git branch
```

Lists all local branches in the current repository

```
$ git branch [branch-name]
```

Creates a new branch

```
$ git checkout [branch-name]
```

Switches to the specified branch and updates the working directory

Final Word

- Cheat sheet is always handy…