SOFTWARE VERSION CONTROL

WHAT IS VERSION CONTROL?

The Need for Version Control

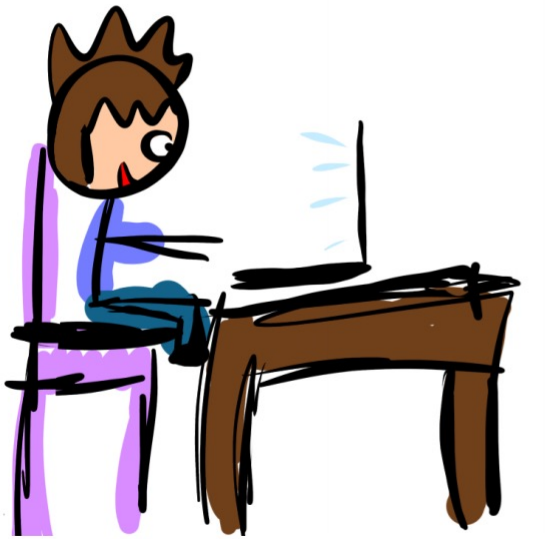
How do I make incremental changes and share my work with others?

How do I go back to the version of this file from (yesterday, last week, last year, ...)?

What changed between version X and version Y of a file?

People have been making changes to the same file (or set of files)... How do I reconcile and merge all these changes?

What is Version Control?



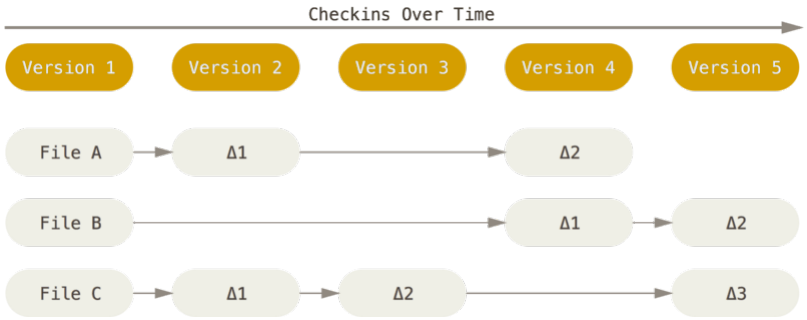
- A system that manages changes to a set files in order to keep a history of changes
- Version Control is similar to:
  - Snapshots of VMs
  - Incremental backups of files
  - Wiki versioning
- When you make a mistake or want to do some experimenting, you can do that in a safe way.

WHAT IS GIT?

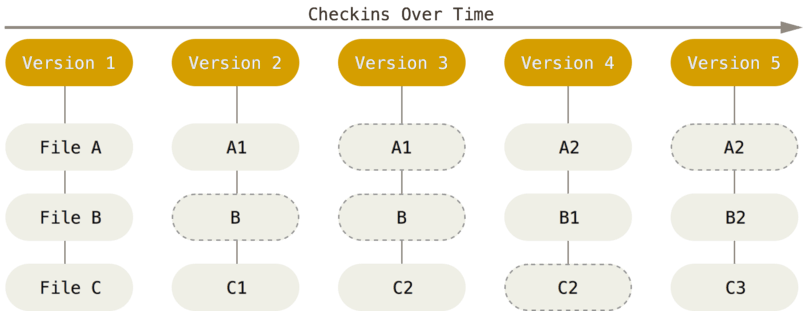
Git

- An open source distributed version control system
- Designed with performance, security and flexibility in mind
- Stores snapshots of the full file instead of diffs
  - Changes are stored in trees
  - Trees contain changed files
  - Commits contain trees

Snapshots, Not Differences

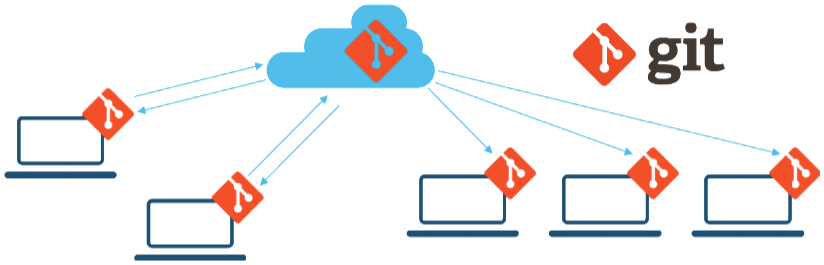


delta-based version control



GIT

Git vs. GitHub



Git is an open source Distributed Version Control System

GitHub is a commercial company, that runs GitHub.com based on Git Version Control System

FIRST-TIME GIT SETUP

How to I get started

1. Install Git
https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

2. Configure your identity on Git

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
$ git config --list
```

GETTING AND CREATING PROJECTS

Initializing a Repository in an Existing Directory

- If you have a project directory that is currently not under version control and you want to start controlling it with Git, you first need to go to that project's directory.

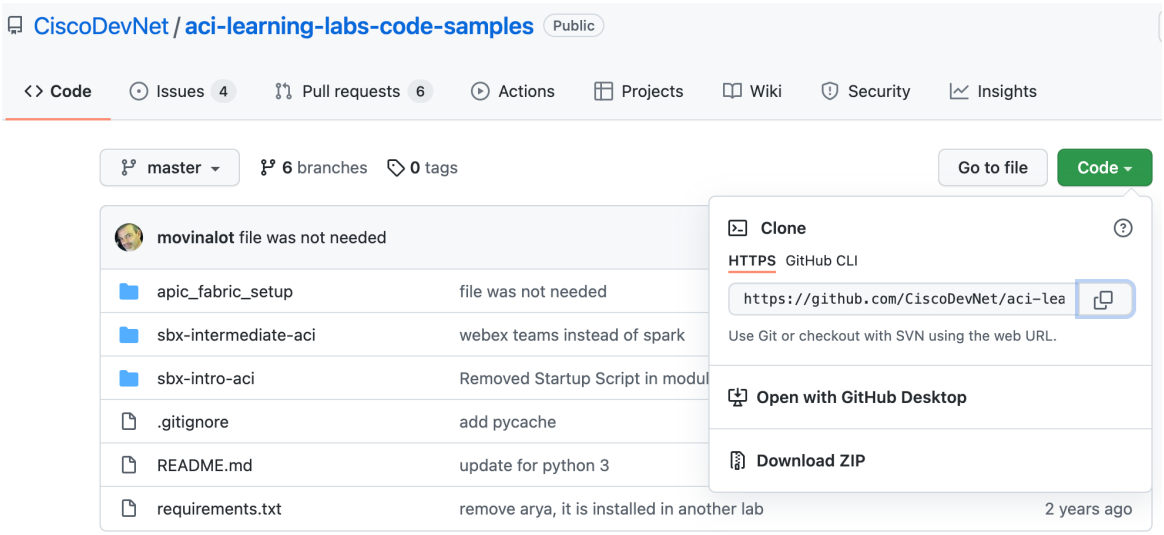
```
$ cd /home/user/my_project
$ git init
```

- This creates a new subdirectory named .git that contains all of your necessary repository files — a Git repository skeleton. At this point, nothing in your project is tracked yet.

Cloning an Existing Repository

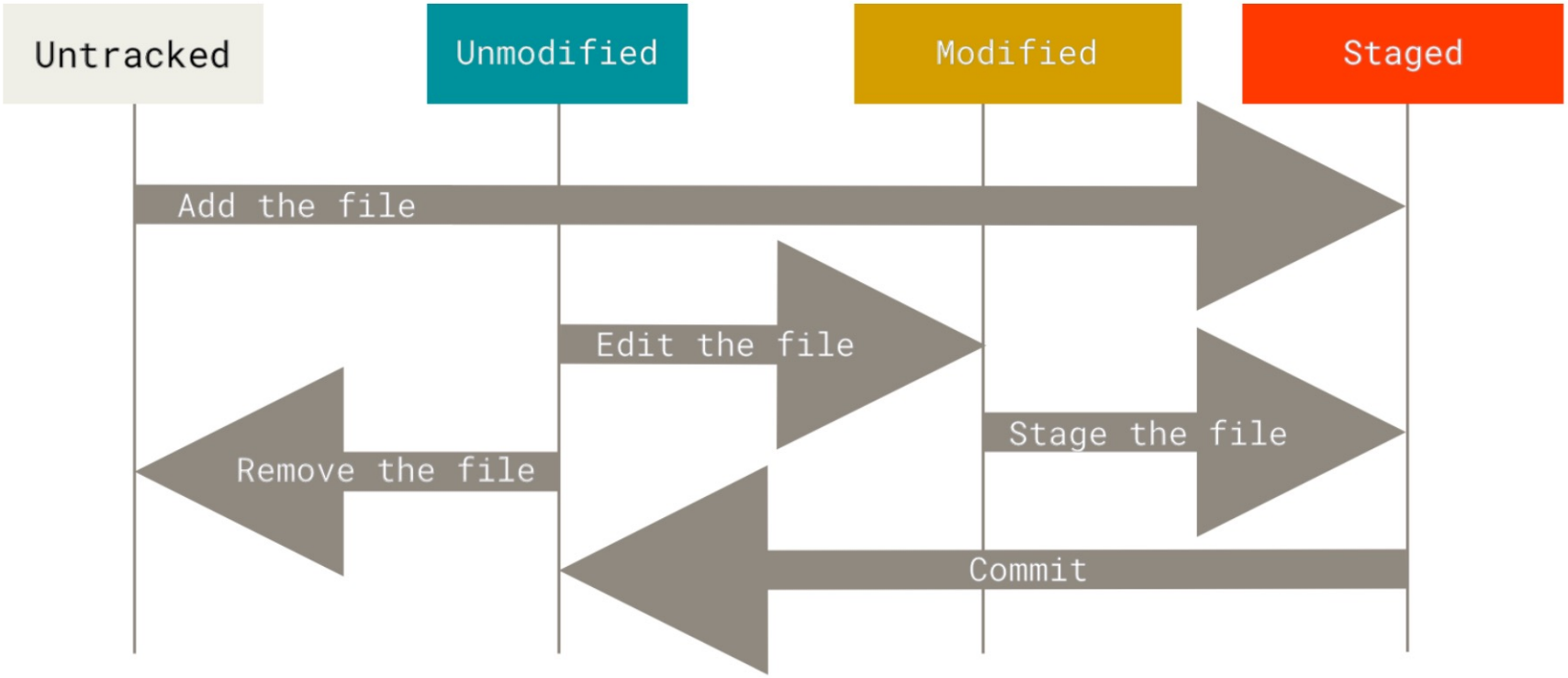
- If you want to get a copy of an existing Git repository — for example, a project you'd like to contribute to — the command you need is git clone

```
$ git clone https://github.com/CiscoDevNet/aci-learning-labs-code-samples.git
```



RECORDING CHANGES TO THE REPOSITORY

The lifecycle of the status of your files



Checking the Status of Your Files

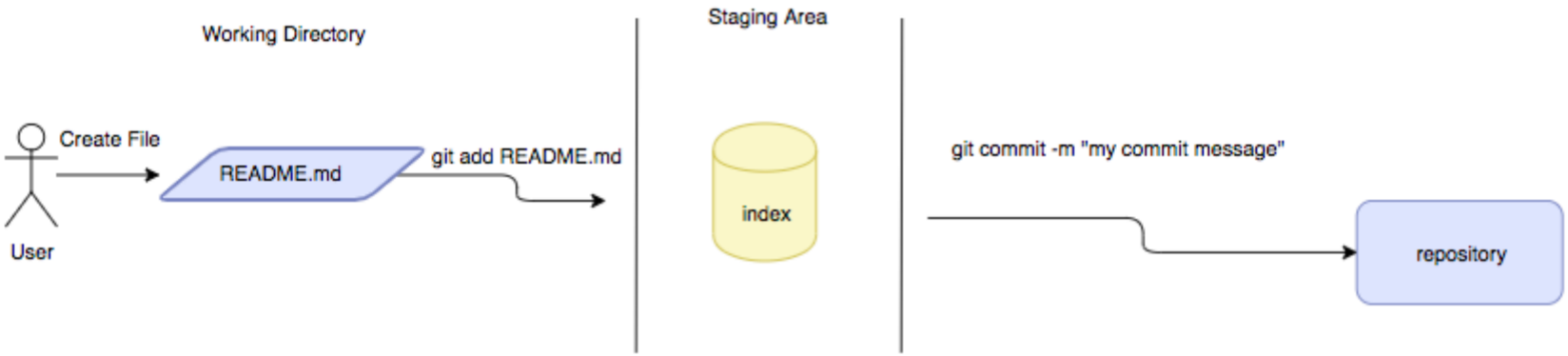
- Git status command show the list of paths/file that have differences between the local directory and the last commit in the repository

```
$ git status
```

- Also shows a list of untracked files that are not part of the repository

```
$$ cd project1
$ touch app.py
$ git status
On branch master
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
app.py
nothing added to commit but untracked files present (use "git add" to track)
```

Adding/Updating file(s) in repo



Tracking New Files

- In order to begin tracking a new file, you use the command git add. To begin tracking the app.py file, you can run this:

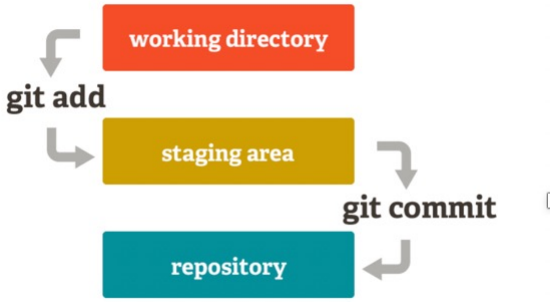
```
$ git add app.py
```

- Add files with a wildcard

```
$ git add a*.py
```

- Add all files that have been changed

```
$ git add .
```



https://git-scm.com/images/about/index1@2x.png

Tracking New Files

```
$ git status
On branch master

No commits yet

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

Staging Modified Files

```
$ git status
On branch master

No commits yet

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

Changes not staged for commit:
  (use "git add <file>..." to update what will be
committed)
  (use "git restore <file>..." to discard changes in
working directory)
        modified:   app.py
```

```
$ git add app.py
$ git status
On branch master

No commits yet

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

If you modify a file after you run git add, you have to run git add again to stage the latest version of the file

Viewing Your Staged and Unstaged Changes

- To see what you've changed but not yet staged, type git diff with no other arguments

```
$ git diff
diff --git a/app.py b/app.py
index 1f7d4f7..9e5e9f1 100644
--- a/app.py
+++ b/app.py
@@ -1,2 +1,2 @@
-def build_config:
+def build_config(params)
     return
$
```

- If you want to see what you've staged that will go into your next commit, you can use git diff --staged. This command compares your staged changes to your last commit:

```
$ git diff --staged
diff --git a/app.py b/app.py
new file mode 100644
index 0000000..1f7d4f7
--- /dev/null
+++ b/app.py
@@ -0,0 +1,2 @@
+def build_config:
+     return
```

Committing Your Changes

- Remember that anything that is still unstaged — any files you have created or modified that you haven't run git add on since you edited them — won't go into this commit. They will stay as modified files on your disk.
- Every time you perform a commit, you're recording a snapshot of your project that you can revert to or compare to later.

```
$ git commit -m "initial version"
[master (root-commit) 4700147] initial version
 1 file changed, 2 insertions(+)
 create mode 100644 app.py
```

Removing Files

- To remove a file from Git, you have to remove it from your tracked files (more accurately, remove it from your staging area) and then commit. The git rm command does that, and also removes the file from your working directory so you don't see it as an untracked file the next time around.

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

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

```
$ git rm app.py
rm 'app.py'
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    app.py
$ git commit -m "deleting app.py"
[master 4ca4881] deleting app.py
 1 file changed, 2 deletions(-)
 delete mode 100644 app.py
```

Viewing the Commit History

- After you have created several commits, or if you have cloned a repository with an existing commit history, you'll probably want to look back to see what has happened. The most basic and powerful tool to do this is the git log command.

```
$ git log
commit
4ca488171144265134db8f907d525e1116a25c4f
(HEAD -> master)
Author: devnet <devnet.cisco.vn@gmail.com>
Date:   Sat Apr 2 11:19:33 2022 -0400

    deleting app.py

commit
47001474f2613cace52f9fca24c42ebca4241c17
Author: devnet <devnet.cisco.vn@gmail.com>
Date:   Sat Apr 2 11:07:55 2022 -0400

    initial version
```

Common options to git log

Option	Description
`-p`	Show the patch introduced with each commit.
`--stat`	Show statistics for files modified in each commit.
`--shortstat`	Display only the changed/insertions/deletions line from the –stat command.
`--name-only`	Show the list of files modified after the commit information.
`--name-status`	Show the list of files affected with added/modified/deleted information as well.
`--abbrev-commit`	Show only the first few characters of the SHA-1 checksum instead of all 40.
`--relative-date`	Display the date in a relative format (for example, “2 weeks ago”) instead of using the full date format.
`--graph`	Display an ASCII graph of the branch and merge history beside the log output.
`--pretty`	Show commits in an alternate format. Option values include oneline, short, full, fuller, and format (where you specify your own format).
`--oneline`	Shorthand for `--pretty=oneline --abbrev-commit` used together.

```
$ git log --oneline
4ca4881 (HEAD -> master) deleting app.py
4700147 initial version
```

git log --pretty=format

Specifier	Description of Output
%H	Commit hash
%h	Abbreviated commit hash
%T	Tree hash
%t	Abbreviated tree hash
%P	Parent hashes
%p	Abbreviated parent hashes
%an	Author name
%ae	Author email
%ad	Author date (format respects the --date=option)
%ar	Author date, relative
%cn	Committer name
%ce	Committer email
%cd	Committer date
%cr	Committer date, relative
%s	Subject

```
$ git log --pretty=format:"%h - %an, %ar : %s"
4ca4881 - devnet, 12 hours ago : deleting app.py
4700147 - devnet, 13 hours ago : initial version
```

Limiting Log Output

Option	Description
`-<n>`	Show only the last n commits
`--since`, `--after`	Limit the commits to those made after the specified date.
`--until`, `--before`	Limit the commits to those made before the specified date.
`--author`	Only show commits in which the author entry matches the specified string.
`--committer`	Only show commits in which the committer entry matches the specified string.
`--grep`	Only show commits with a commit message containing the string
`-S`	Only show commits adding or removing code matching the string

git show

```
$ git show 4700147
commit 47001474f2613cace52f9fca24c42ebca4241c17
Author: devnet <devnet.cisco.vn@gmail.com>
Date:   Sat Apr 2 11:07:55 2022 -0400

    initial version

diff --git a/app.py b/app.py
new file mode 100644
index 0000000..1f7d4f7
--- /dev/null
+++ b/app.py
@@ -0,0 +1,2 @@
+def build_config:
+    return
```

UNDOING THINGS

Redo a commit

```
$ git commit -m 'Initial commit'
$ git add forgotten_file
$ git commit --amend
```

Unstaging a Staged File with git restore

```
$ git add .
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   CONTRIBUTING.md
        renamed:    readme.md -> readme
```

```
$ git restore --staged CONTRIBUTING.md
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    readme.md -> readme

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

Unmodifying a Modified File with git restore

- It's important to understand that git restore -- <file> is a dangerous command. Any local changes you made to that file are gone — Git just replaced that file with the last staged or committed version. Don't ever use this command unless you absolutely know that you don't want those unsaved local changes, anything you lose that was never committed is likely never to be seen again.

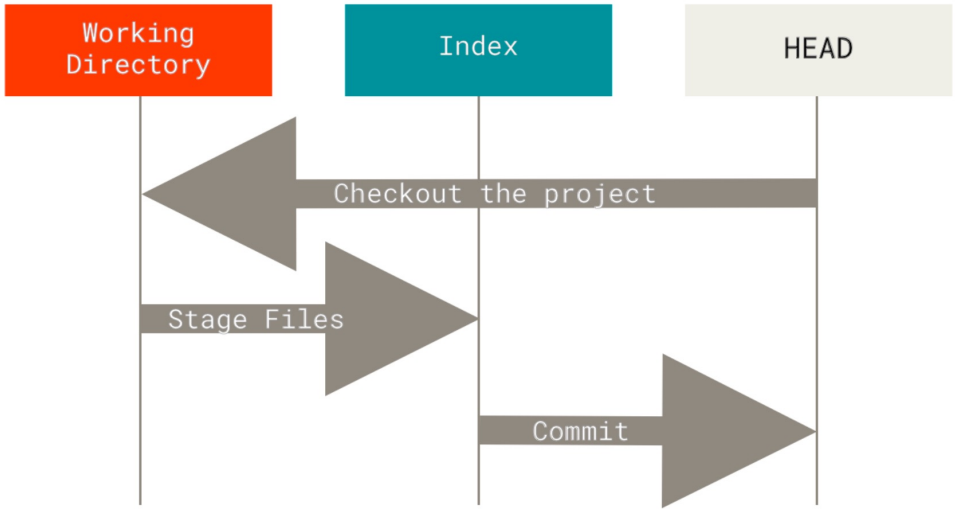
- If you would like to keep the changes you've made to that file but still need to get it out of the way for now, Git Branching is a generally better way to go.

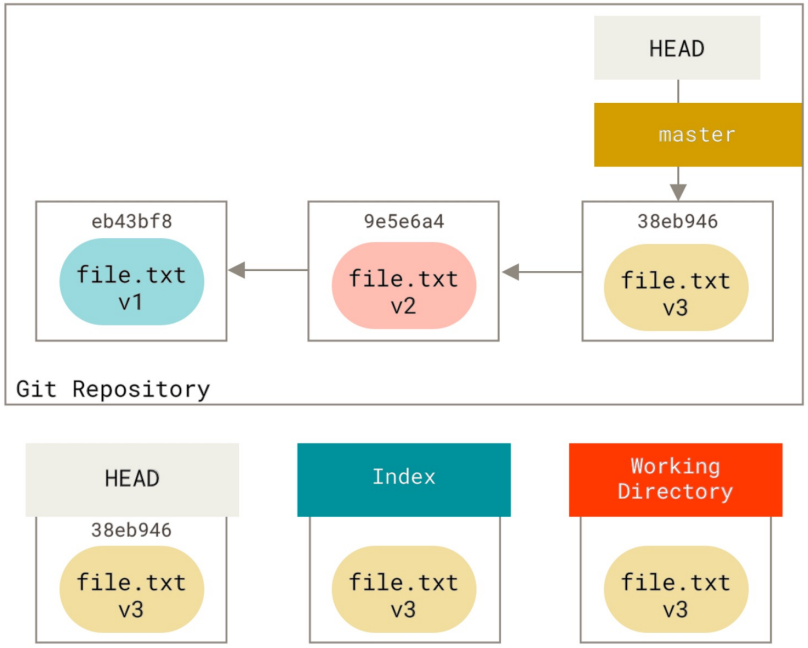
```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   CONTRIBUTING.md
no changes added to commit (use "git add" and/or "git commit -a")
```

```
$ git restore CONTRIBUTING.md
$ git status
On branch master
nothing to commit, working tree clean
```

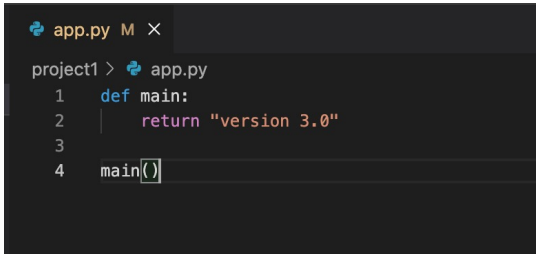
RESET DEMYSTIFIED

The Workflow





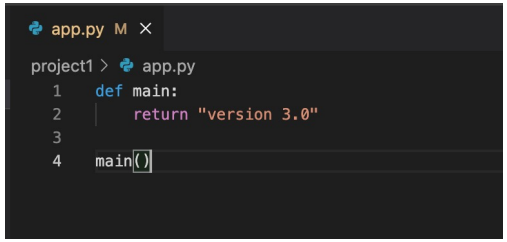
Step 1: Move HEAD




```
$ git log --oneline
b85eaca (HEAD -> master) version 3.0
4e54fbd version 2.0
59f0a11 version 1.0
4ca4881 deleting app.py
4700147 initial version
```

```
$ git reset 59f0a11 --soft
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   app.py

$ git log --oneline
59f0a11 (HEAD -> master) version 1.0
4ca4881 deleting app.py
4700147 initial version
```

Step 2: Updating the Index (--mixed)




```
$ git log --oneline
31ce232 (HEAD -> master) version 3.0
053fb05 version 2.0
59f0a11 version 1.0
4ca4881 deleting app.py
4700147 initial version
```

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

no changes added to commit (use "git add" and/or "git
commit -a")
$ git log --oneline
59f0a11 (HEAD -> master) version 1.0
4ca4881 deleting app.py
4700147 initial version
```

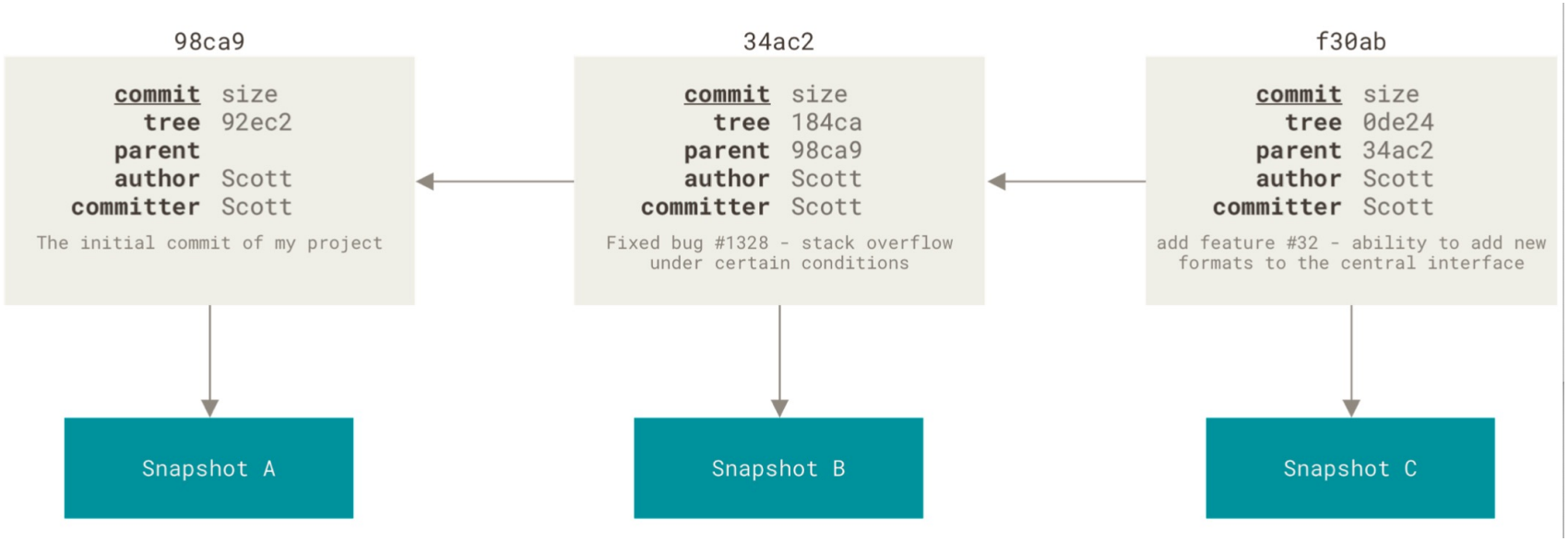
Step 3: Updating the Working Directory (--hard)


```
$ git log --oneline
433d7fc (HEAD -> master) version 3.0
bbaf481 version 2.0
59f0a11 version 1.0
4ca4881 deleting app.py
4700147 initial version
```

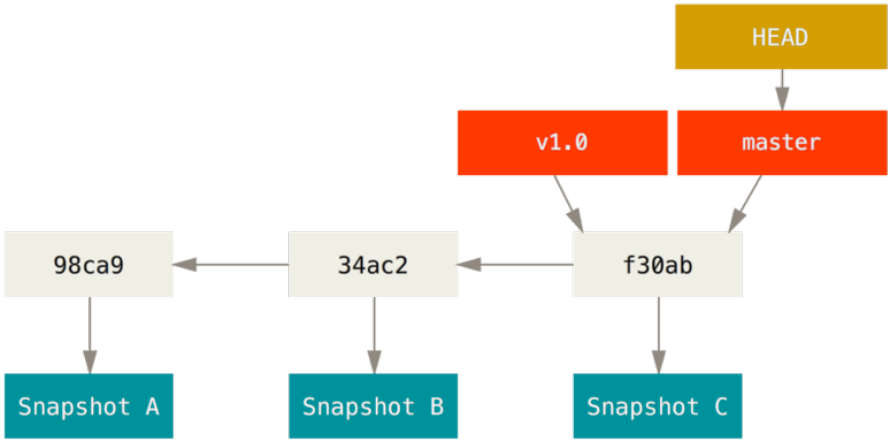
```
$ git reset 59f0a11 --hard
HEAD is now at 59f0a11 version 1.0
$ git status
On branch master
nothing to commit, working tree clean
$ git log --oneline
59f0a11 (HEAD -> master) version 1.0
4ca4881 deleting app.py
4700147 initial version
```

BRANCHING AND MERGING

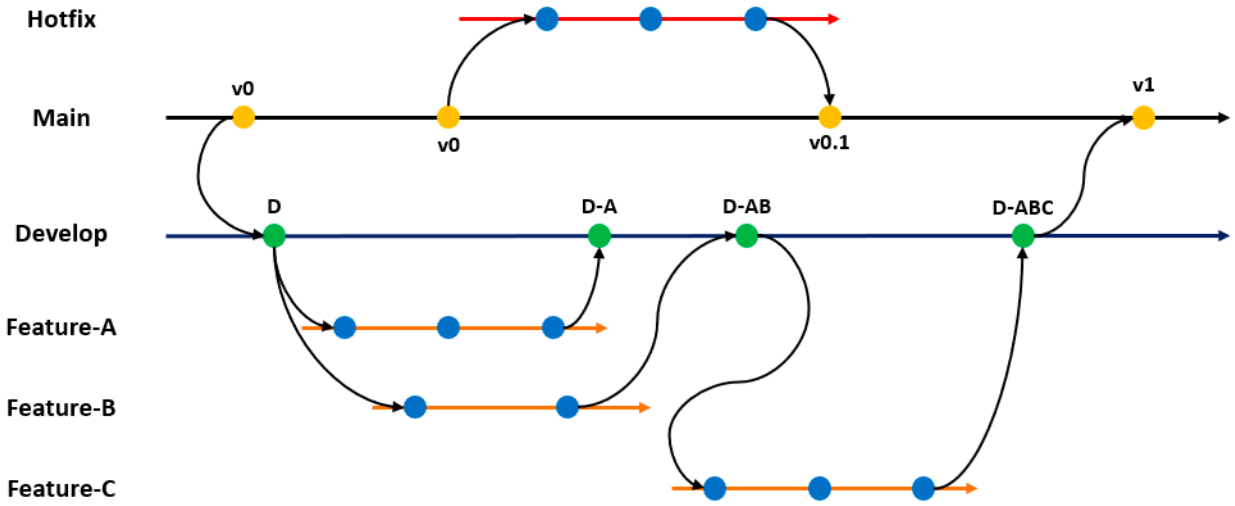
Commits and their parents



Git branch

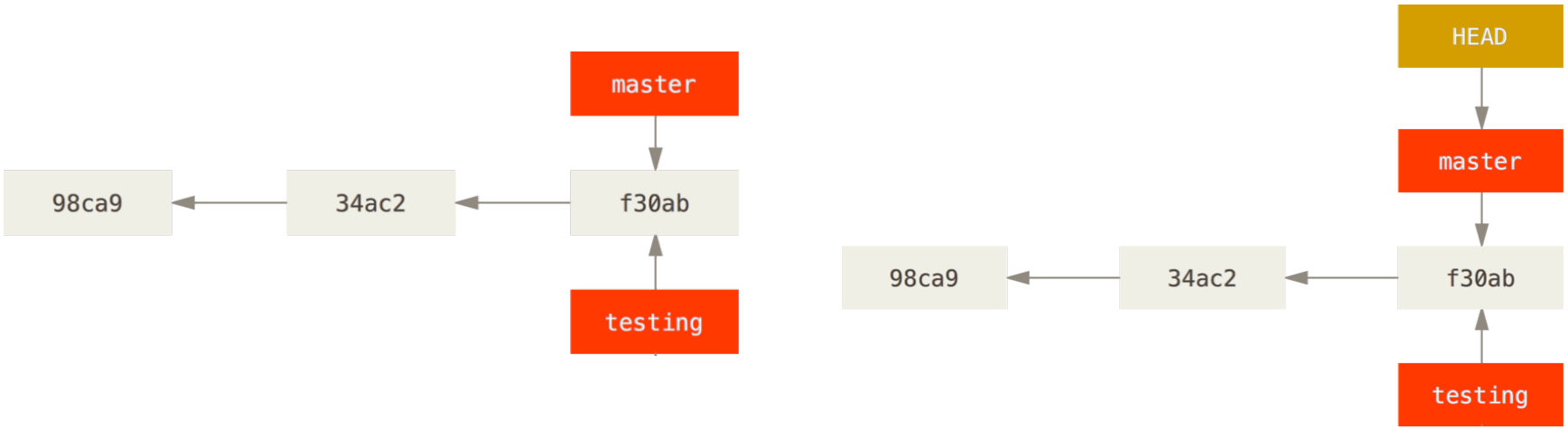


A branch in Git is simply a lightweight movable pointer to one of these commits. The default branch name in Git is master. As you start making commits, you're given a master branch that points to the last commit you made. Every time you commit, the master branch pointer moves forward automatically.



Git branch - Creating a New Branch

```
$ git branch testing
```



The git branch command only **created** a new branch — it didn't switch to that branch.

git branch

- Create a branch in your repository

```
$ git branch <branch name>
```

- Delete a branch in your repository

```
$ git branch –d <branch name>
```

git branch

- A branch:
  - Creates an independent path with a copy of the code
  - Diverges from a commit
  - Allows usage of git functionality without affecting the original line
  - Can be merged back to the original line when ready (git merge)
  - Can be deleted and never merged back to the original line
- List all of the branches in your repository

```
$ git branch
```

```
$ git branch testing
$ git branch
* master
  testing
```

git checkout

- Switch between branches
- Make <branch name> the working branch

```
$ git checkout <branch name>
```

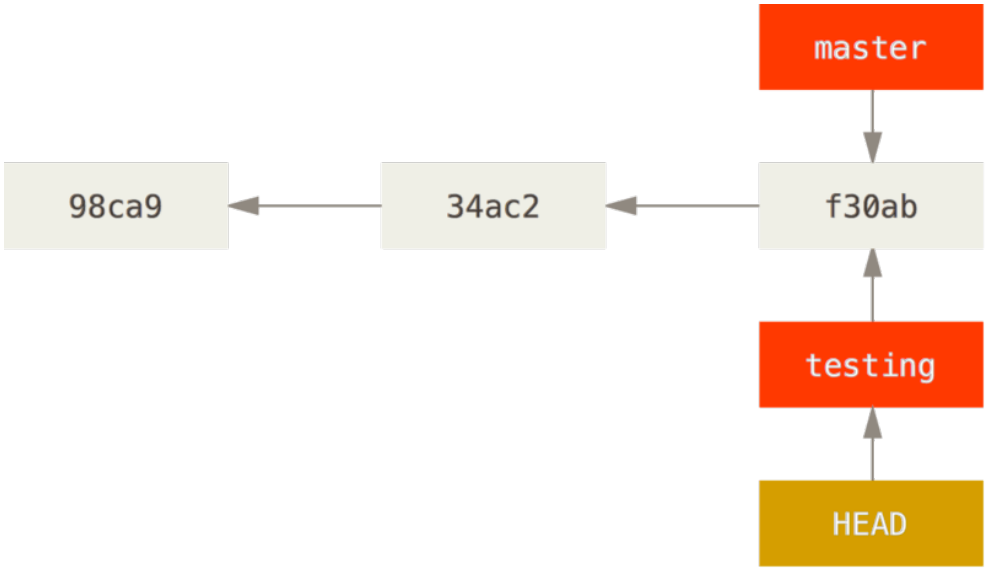
- Create a branch and make it the working branch

```
$ git checkout -b <branch name>
```

git checkout - Switching Branches

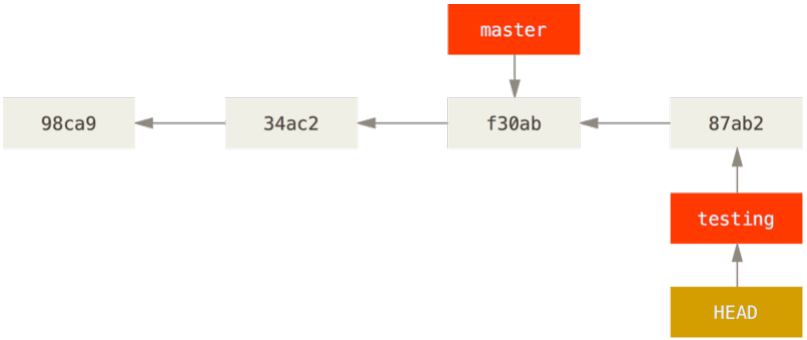
```
$ git checkout testing
```

This moves HEAD to point to the testing branch.

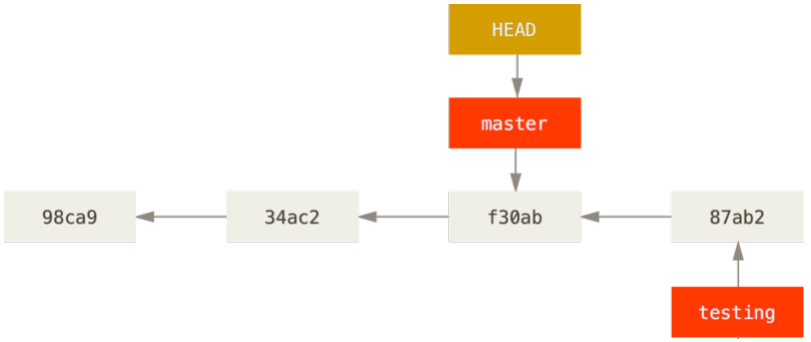



```
$ ls
app.py  CONTRIBUTING.md  readme
$ git status
On branch testing
nothing to commit, working tree clean
$ git branch
  master
  testing

$ git checkout testing
Switched to branch 'testing'
```

git checkout - Switching Branches

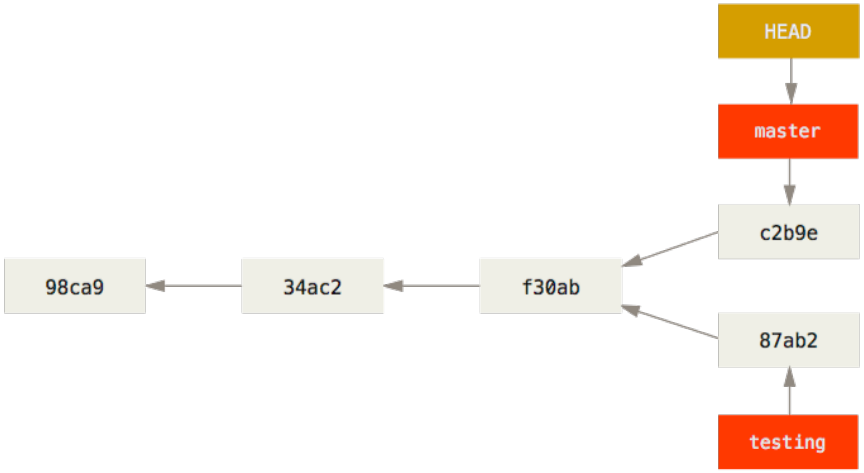



```
$ touch test.py
$ git add test.py
$ git commit -m "made other change”
$ git log --oneline
338f58f (HEAD -> testing) made other
change
59f0a11 (master) version 1.0
4ca4881 deleting app.py
4700147 initial version
```




```
$ git switch master
Switched to branch 'master'
$ git log --oneline
59f0a11 (HEAD -> master) version 1.0
4ca4881 deleting app.py
4700147 initial version
```

Divergent history




```
$ git add app.py
$ git commit -m "version 2.0"
[master c51d77a] version 2.0
 1 file changed, 1 insertion(+), 1 deletion(-)
```

```
$ git log --oneline --decorate --graph --all
* c51d77a (HEAD -> master) version 2.0
| * 338f58f (testing) made other change
|/
* 59f0a11 version 1.0
* 4ca4881 deleting app.py
* 4700147 initial version
```

Note

- From Git version 2.23 onwards you can use git switch instead of git checkout to:
  - Switch to an existing branch: git switch testing-branch.
  - Create a new branch and switch to it: git switch -c new-branch. The -c flag stands for create, you can also use the full flag: --create.
  - Return to your previously checked out branch: git switch -.

git merge

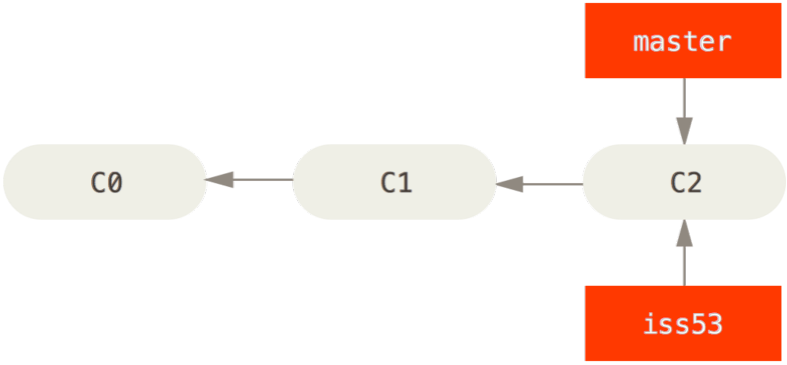
- Join two or more branches together
- You must be on the branch you want to merge INTO when you execute these command (e.g. master)
- Merge a branch on top of the current branch

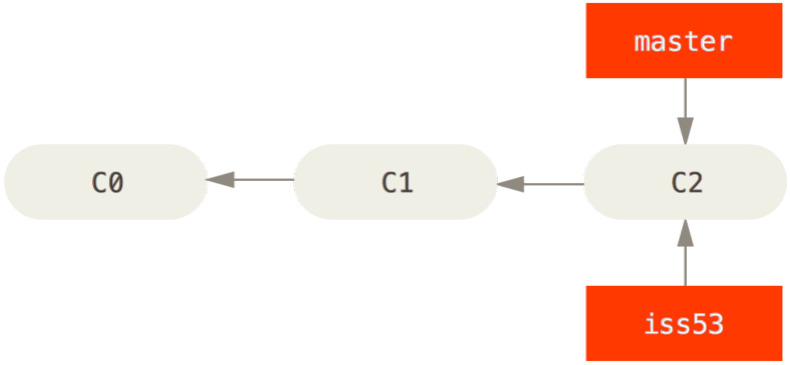
```
$ git merge <branch name to merge from>
```

- Merge more than one branch on top of the current branch

```
$ git merge <1st branch name> <2nd branch name>
```

git merge - “fast-forward”






```
$ git checkout -b iss53
Switched to a new branch "iss53"
```

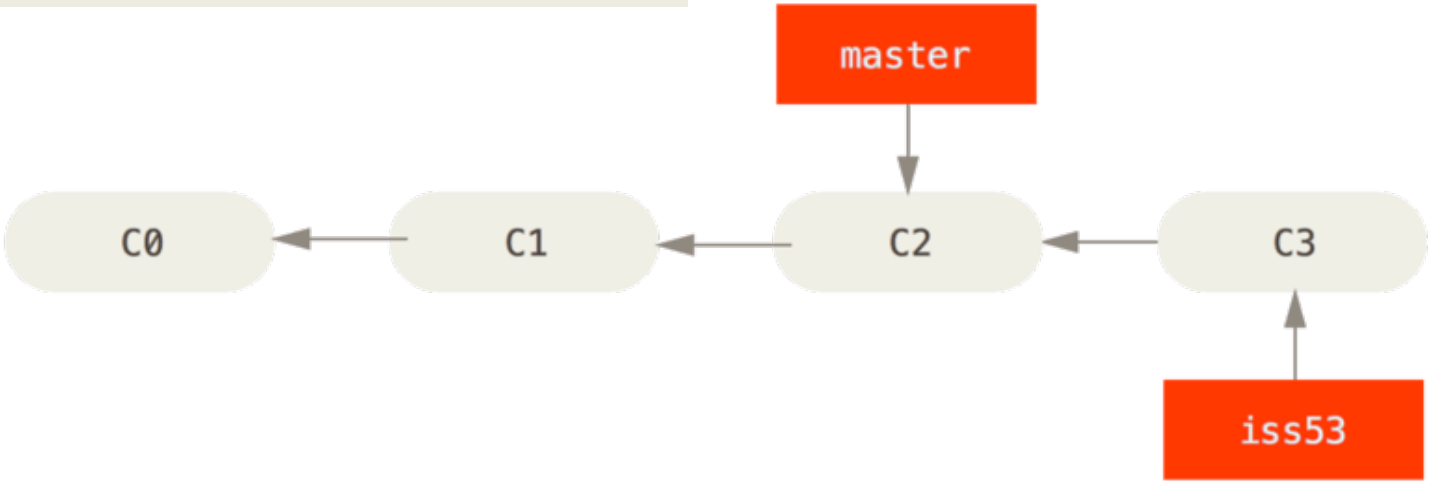
This is shorthand for:

```
$ git branch iss53
$ git checkout iss53
```

git merge - “fast-forward”

```
$ git switch iss53
$ touch iss53.txt
$ git add .
$ git commit -m "Creating iss53.txt file"
[iss53 ef6c926] Creating iss53.txt file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 iss53.txt
```

```
$ git log --oneline
ef6c926 (HEAD -> iss53) Creating iss53.txt file
22f82eb Issue 53
338f58f (testing) made other change
59f0a11 version 1.0
4ca4881 deleting app.py
4700147 initial version
```

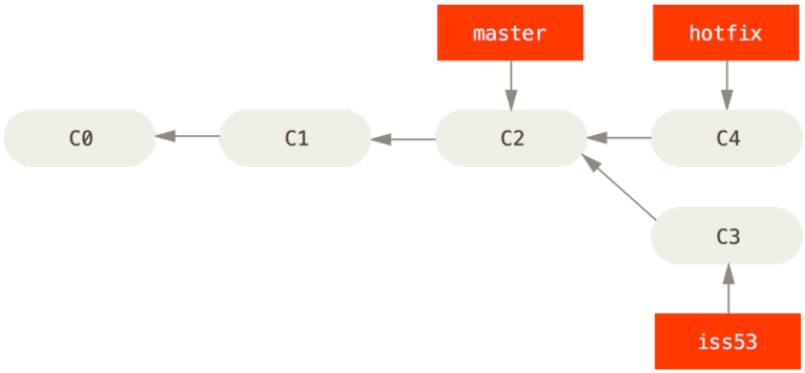


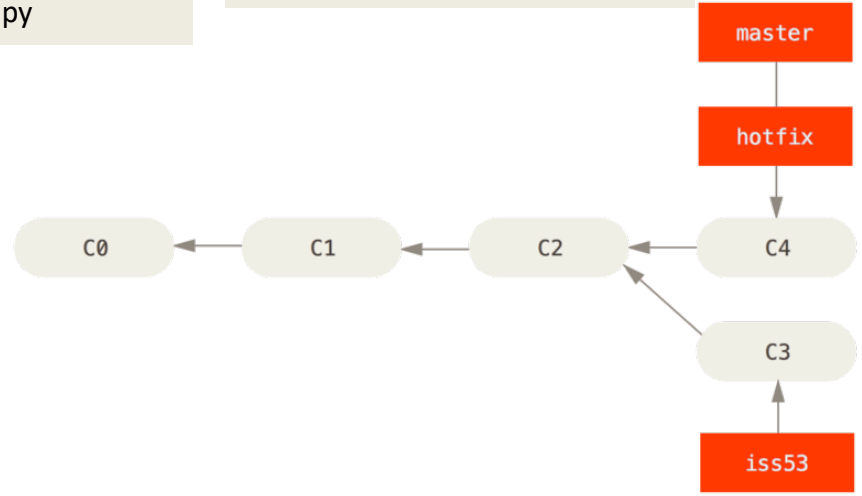
git merge - “fast-forward”

```
$ git switch master
Switched to branch 'master'
$ ls
app.py  CONTRIBUTING.md  readme
```

```
$ git checkout -b hotfix
Switched to a new branch 'hotfix'
$ touch hotfix.py
$ git add hotfix.py
$ git commit -m "Adding hotfix.py"
[hotfix e1eb79d] Adding hotfix.py
 1 file changed, 0 insertions(+), 0
deletions(-)
 create mode 100644 hotfix.py
```

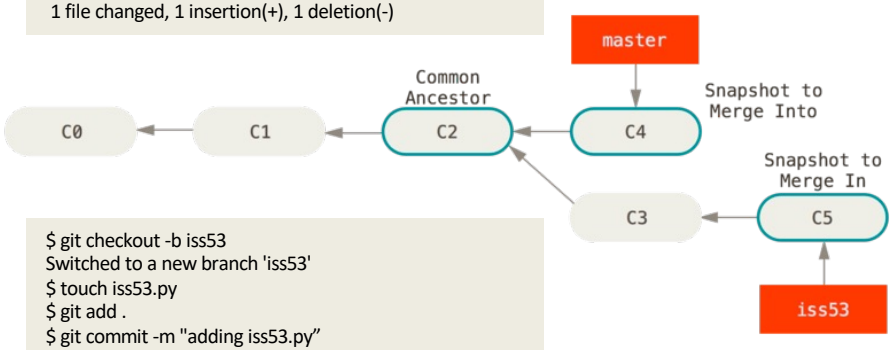
```
$ git checkout master
$ git merge hotfix
Updating c51d77a..e1eb79d
Fast-forward
 hotfix.py | 0
 1 file changed, 0 insertions(+), 0
deletions(-)
 create mode 100644 hotfix.py
```





Git merge – basic merging

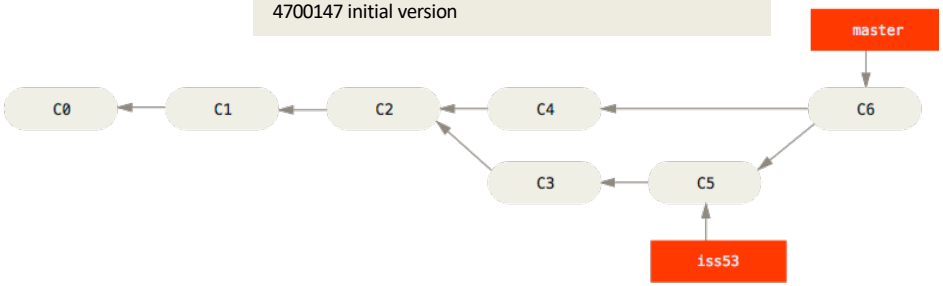
```
$ git log --oneline
59f0a11 (HEAD -> master) version 1.0
4ca4881 deleting app.py
4700147 initial version
$ git add app.py
$ git commit -m "version 2.0"
[master 2479b53] version 2.0
 1 file changed, 1 insertion(+), 1 deletion(-)
```




```
$ git checkout -b iss53
Switched to a new branch 'iss53'
$ touch iss53.py
$ git add .
$ git commit -m "adding iss53.py”
$ git add .
$ git commit -m "iss53.py version 2.0"
```

View before a merge

```
$ git merge iss53
Merge made by the 'recursive' strategy.
 iss53.py | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 iss53.py
$ git log --oneline
600f337 (HEAD -> master) Merge branch 'iss53'
2479b53 version 2.0
b3c4940 (iss53) iss53.py version 2.0
f3a0e2f adding iss53.py
59f0a11 version 1.0
4ca4881 deleting app.py
4700147 initial version
```



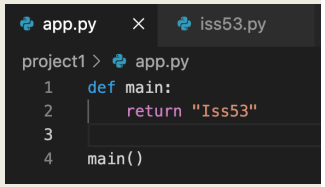
View after the merge

Basic Merge Conflicts (1)

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

no changes added to commit (use "git add" and/or "git commit -
a")
$ git diff
diff --git a/app.py b/app.py
index 225d579..797c179 100644
--- a/app.py
+++ b/app.py
@@ -1,4 +1,4 @@
 def main:
-    return "version 2.0"
+    return "version 3.0"

 main()
\ No newline at end of file

$ git add .
$ git commit -m "version 3.0"
[master 3ad1161] version 3.0
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Master branch

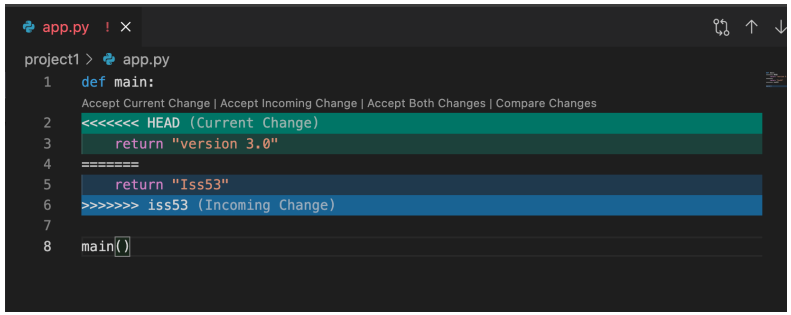
```
$ git switch iss53
Switched to branch 'iss53'

$ git status
On branch iss53
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working
directory)
        modified:   app.py

no changes added to commit (use "git add" and/or "git commit -
a")
$ git diff
diff --git a/app.py b/app.py
index 45ec2b8..03d600a 100644
--- a/app.py
+++ b/app.py
@@ -1,4 +1,4 @@
 def main:
-    return "version 1.0"
+    return "Iss53"

 main()
\ No newline at end of file
$ git add .
$ git commit -m "Iss53"
[iss53 b8c8677] Iss53
 1 file changed, 1 insertion(+), 1 deletion(-)
```

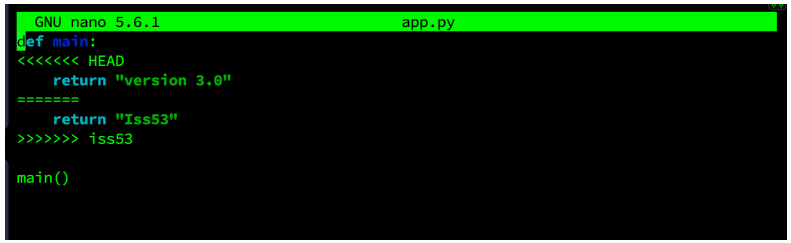


Iss53 branch

Basic Merge Conflicts (1)

```
$ git switch master
Switched to branch 'master'
$ git merge iss53
Auto-merging app.py
CONFLICT (content): Merge conflict in app.py
Automatic merge failed; fix conflicts and then commit the result.
```






```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   app.py

no changes added to commit (use "git add" and/or "git commit -
a")
$ git add app.py
$ git commit -m "merging iss53 branch"
[master 3fe2664] merging iss53 branch
```

WORKING WITH REMOTES

Connecting to GitHub with SSH

Checking for existing SSH keys

1. Open Terminal.
2. Enter `ls -al ~/.ssh` to see if existing SSH keys are present:

```
$ ls -al ~/.ssh
# Lists the files in your .ssh directory, if they exist
```

3. Check the directory listing to see if you already have a public SSH key. By default, the filenames of the public keys are one of the following:
   - *id_rsa.pub*
   - *id_ecdsa.pub*
   - *id_ed25519.pub*

https://docs.github.com/en/free-pro-team@latest/github/authenticating-to-github/connecting-to-github-with-ssh

Generating a new SSH key

1. Open Terminal.
2. Paste the text below, substituting in your GitHub email address.

```
$ ssh-keygen -t ed25519 -C "your_email@example.com"
```

> **Note:** If you are using a legacy system that doesn't support the Ed25519 algorithm, use:
>
> `$ ssh-keygen -t rsa -b 4096 -C "your_email@example.com"`

This creates a new ssh key, using the provided email as a label.

```
> Generating public/private ed25519 key pair.
```

3. When you're prompted to "Enter a file in which to save the key," press Enter. This accepts the default file location.

```
> Enter a file in which to save the key (/Users/you/.ssh/id_ed25519): [Press ent
```

4. At the prompt, type a secure passphrase. For more information, see "Working with SSH key passphrases".

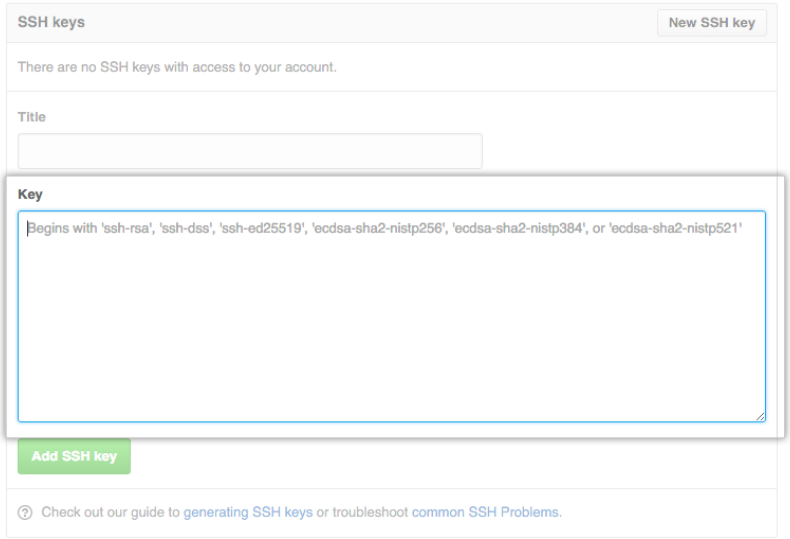
```
> Enter passphrase (empty for no passphrase): [Type a passphrase]
> Enter same passphrase again: [Type passphrase again]
```

Connecting to GitHub with SSH

Adding a new SSH key to your GitHub account

locate the hidden .ssh folder, open the file in your favorite text editor, and copy it to your clipboard.

```
osboxes@ubuntu:~/Documents/git/git-intro$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQCwWQ3t77WoGUH20oWCUE71f3AdWOdyaHFQVSS7S4
87wEylcB6hCN14G9KPt+8vXptVNFTyILvm9/vflOMapAKuYbO0yeW0LO9zDXLlS/dTafmgAeVhBsXy
yDCadHhcJGbsDWA+Bj4fMWkdAvJjLpDz8jJoDiCjxwVbZUAa8xFOkrtMoYl8ik4L0lVztk30IqD7RH
jTBRnSSk4AjQD+5udON1ZZNhYd+cUUE1rY5KMdw4AUXpaInno1Vee3YJWtmM3XY0gduKjLddmSxNY5
/d2u76oasGOxMv9UGWNybX9larkLDLbx6DKgWLB0NQ3ddQoT2VH5LgvKBz3CjmAcuP4axYLmt/Y6yI
kyXcgQldHsrEKLYpLj7Joxe7+N2PyUzHAhXP50fy7a/8WI+laJPgaoPjho2KPky/v1bSN2tutrHRjT
kV6uLCFtFwkIUUirswLly/UenC+0Hy5J/nxfUw== sinhhoang210@gmail.com
osboxes@ubuntu:~/Documents/git/git-intro$
```



Connecting to GitHub with SSH

1. Open Terminal.
2. Enter the following:

```
$ ssh -T git@github.com
# Attempts to ssh to GitHub
```

You may see a warning like this:

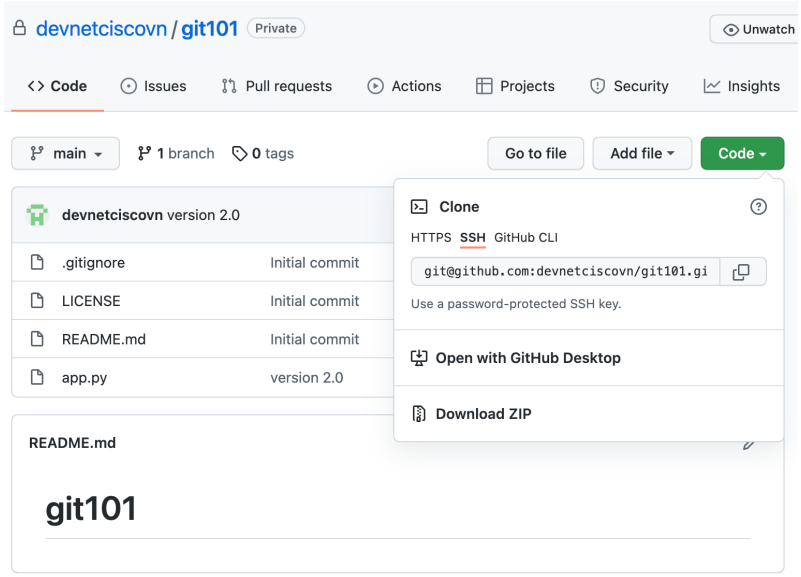
```
> The authenticity of host 'github.com (IP ADDRESS)' can't be established.
> RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.
> Are you sure you want to continue connecting (yes/no)?
```

3. Verify that the fingerprint in the message you see matches one of the messages in step 2, then type `yes`:

```
> Hi username! You've successfully authenticated, but GitHub does not
> provide shell access.
```

$ ssh -T git@github.com
The authenticity of host 'github.com (20.205.243.166)' can't be established.
ECDSA key fingerprint is SHA256:p2QAMXNIC1TJYWeIOttrVc98/R1BUFWu3/LiyKgUfQM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com,20.205.243.166' (ECDSA) to the list of known hosts.
Hi devnetciscovn! You've successfully authenticated, but GitHub does not provide shell access.

Connecting to GitHub with SSH




```
$ ssh -T git@github.com
The authenticity of host 'github.com (20.205.243.166)' can't be established.
ECDSA key fingerprint is SHA256:p2QAMXNIC1TJYWeIOttrVc98/R1BUFWu3/LiyKgUfQM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com,20.205.243.166' (ECDSA) to the list of known hosts.
Hi devnetciscovn! You've successfully authenticated, but GitHub does not provide shell access.
```

git remote

Adding Remote Repositories

git remote add <shortname> <url>:

$ git remote add pb https://github.com/paulboone/ticgit

Renaming and Removing Remotes

git remote rename

git remote remove/git remote rm

$ git remote rename pb paul

$ git remote remove paul

git remote

Showing Your Remotes:

git remote –v

```
$ git remote -v
origin  git@github.com:devnetciscovn/git101.git (fetch)
origin  git@github.com:devnetciscovn/git101.git (push)
```

Inspecting a Remote

it remote show <remote>

$ git remote show origin

```
$ git remote show origin
* remote origin
  Fetch URL: git@github.com:devnetciscovn/git101.git
  Push  URL: git@github.com:devnetciscovn/git101.git
  HEAD branch: main
  Remote branch:
    main tracked
  Local branch configured for 'git pull':
    main merges with remote main
  Local ref configured for 'git push':
    main pushes to main (up to date)
```

Fetching and Pulling from Your Remotes

git fetch <remote>

git merge origin/main

```
$ git fetch origin
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 687 bytes | 687.00 KiB/s, done.
From github.com:devnetciscovn/git101
   cdab867..034af3c  main       -> origin/main

$ git merge origin/main
Updating cdab867..034af3c
Fast-forward
 app.py | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 app.py
$
$ git log --oneline
034af3c (HEAD -> main, origin/main, origin/HEAD) Version 1.0
cdab867 Initial commit
```

Pushing to Your Remotes

git push <remote> <local_branch>

```
$ git push origin
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 289 bytes | 144.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:devnetciscovn/git101.git
   034af3c..f1ab206  main -> main
```

git pull

- Apply the latest changes from the repository/branch that you cloned from to your local repository that you are currently on

```
$ git pull
```

- Apply the latest changes from the specified repository/branch to your local repository that you are currently on

```
$ git pull origin <branch name>
```