CHAPTER 1

Intro to Git & Git Basics

Sharif University of Technology
Computer Engineering Department
Presented By S. M. Masoud Sadrnezhaad
Source: Michael Koby
What We’ll Cover

- What is Version Control
- Why You Should Use Version Control
- Types of Version Control
- Overview of How Version Control Works
What is Version Control?
...a system that lets you track changes in your source code by “checking in” your code into the system.
Version Control Allows You To

• Keep track of changes you’ve made to a project over time
Version Control Allows You To

- Keep track of changes you’ve made to a project over time
- Create a branch of code to allow you to experiment without affecting your working program
Version Control Allows You To

- Keep track of changes you’ve made to a project over time
- Create a branch of code to allow you to experiment without affecting your working program
- Make collaboration on your projects easier to handle
Version Control Allows You To

- Keep track of changes you’ve made to a project over time
- Create a branch of code to allow you to experiment without effecting your working program
- Make collaboration on your projects easier to handle
- And many other tasks associated with source code (tagging, blaming, release branches, etc)
Why You Should Use Version Control
Why You Should Use Version Control

- Track Your Changes
Why You Should Use Version Control

- Track Your Changes
- Get Back to Working Code More Quickly
Why You Should Use Version Control

- Track Your Changes
- Get Back to Working Code More Quickly
- Easier Collaboration
Why You Should Use Version Control

- Track Your Changes
- Get Back to Working Code More Quickly
- Easier Collaboration
- Easier Backups
Why You Should Use Version Control

- Track Your Changes
- Get Back to Working Code More Quickly
- Easier Collaboration
- Easier Backups
- Sandboxing
Types of Version Control
Types of Version Control

- Centralized
- Distributed
Centralized
Central Version Control Server
Distributed
Available Version Control Systems
Centralized VCS

- CVS
Centralized VCS

- CVS
- Subversion (SVN)
Centralized VCS

- CVS
- Subversion (SVN)
- Team Foundation Server (TFS)
Distributed VCS

- Git
Distributed VCS

- Git
- Mecurial (hg)
Distributed VCS

- Git
- Mecurial (hg)
- Bazaar
Online VCS Hosting
Online VCS Hosting

• Github
Online VCS Hosting

- Github
- Bitbucket
Online VCS Hosting

- Github
- Bitbucket
- Codeplex
How They Work
Next Episode?
CHAPTER 2

Version Control with Git

Intro to Git & Git Basics

Sharif University of Technology
Computer Engineering Department
Presented By S. M. Masoud Sadrnezhaad
Source: Michael Koby
Why Git?
What is Git?
a distributed version control system created by Linus Torvalds, creator of Linux, to replace BitKeeper as the VCS used for maintaining the Linux kernel
Design Goals for Git
Design Goals for Git

- Speed
- Simplicity
Design Goals for Git

• Speed
• Simplicity
• Strong support for non-linear development
• Ability to handle large projects
What Makes Git Different?
Snapshots, Not Differences
It’s All (Mostly) Local
Three Main Stages
Three Main Stages

- Committed
Three Main Stages

- Commited
- Staged
Three Main Stages

- Commited
- Staged
- Modified
Installing Git
Let's Get Started,
Open Your Terminal
Version Control with Git

Branches

Sharif University of Technology
Computer Engineering Department
Presented By S. M. Masoud Sadrnezhaad
Source: Michael Koby
Git commits are pointers to the previous commit.
Going Hands On
Merging
Fast Forward Merge
Version Control With Git
Branches - Part Deux

Sharif University of Technology
Computer Engineering Department
Presented By S. M. Masoud Sadrnezhaad
Source: Michael Koby
Fast Forward Merge

C1 ← C2 ← C3 ← C4

HEAD
Master

(Testing C1)

Testing
Three Way Merge (aka Recursive Merge)
Long Running Branches Stick Around
Topic Branches Center Around Features, Bug Fixes, Etc
Merged Back Into a Long Running Branch
Topic Branches Are Usually Deleted When You’re Done With Them
Deleting a Branch
git branch -D BRANCHNAME
git branch -D testing
Remote Branches
git push REMOTE BRANCHNAME
Deleting Remote Branches
git push REMOTE :BRANCHNAME
git push origin :testing
That's all folks!
CHAPTER 5

Sharif University of Technology
Computer Engineering Department
Presented By S. M. Masoud Sadrnezhaad
Source: Michael Koby
Fast Forward Merge

C1 ────── C2 ────── C3 ────── C4
  |         |         |         |
  │         │         │         │
  │         │         │         │
  HEAD      Master   Testing   Testing C1
Three Way Merge
(aka Recursive Merge)
Merged Without Rebase
3 Way Merge Result
Merged With Rebase
Never rebase a public branch!
Let's see in action
Initiate Git and It's Files
Working With Changes
# make a new directory and go to project path
> mkdir git_repo
> cd git_repo

# initialize git
> git init
Initialized empty Git repository in /path/to/git/repository/.git/
> ls -a
. .. .git

# let's have a deeper look at .git directory
> ls -a .git/
. .. branches config description HEAD hooks info objects refs
# define your name and email address (this config is per machine)
> git config --global user.name "Masoud Sadrnezhaad"
> git config --global user.email smmsadrnezh@gmail.com

# checkout your git config
> git config --list
user.name=Masoud Sadrnezhaad
user.email=smmsadrnezh@gmail.com
credential.helper=cache
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
# create and open README.md file with your default system editor.
> vi README.md

# let's write a readme file (with or without markdown syntax)

# go to insert mode by click "I" one time
# and go back to command mode by Esc. save the file by "wq"

# make sure everything ok.
> ls -a
.  ..  .git  README.md

# check modification and it's level (slide 62) you see modifications at first level.
> git status
nothing added to commit but untracked files present (use "git add" to track...
# level up this file to staged level
> git add README.md

# look at git status. you see changes in staged level.
> git status

# add staged changes to new commit.
# write a meaningful message for your commit.
> git commit -m "added readme file"
[master (root-commit) 021bd57] commit message
  0 files changed
  create mode 100644 README

# look at git status. there is nothing to commit because changes are committed.
> git status
# let's take a look at created commits.
> git log
commit 021bd57955b472d8e6979ba71e4907e9f1e3ab8b
Author: Masoud Sadrnezhaad <smmsadrnezh@gmail.com>
Date: Fri May 1 02:12:46 2015 +0430

commits message

# final notes:

# it's easier to add and commit at once.
> git commit -am "commit message"

# this command add All files in your project directory
> git add -A
Undoing Things
# adding new modification to previous commits.

> git commit --amend

# you see commit message as it's name in first line.  
# uncomment changes you want to amend to that commit  
# then save the file and close the editor.

# you see this modification in your last commit  
> git log
# do some modification and stage them.

# now imagine that you want to unstage one of added files.
# to undo "git add"

```bash
> git reset HEAD README.md
```

# undo to last working version of that specific file.
> git checkout -- README.md

# undo to last working commit.
# actually this one reset everything not only one file.

> git reset --hard

HEAD is now at 11d075a commit message
Branching
# see list of all available branches.
> git branch
* master

# asterix indicates that HEAD is pointing to master branch.

# let's create a new branch and name it dev
> git branch dev

# HEAD is pointing to last commit of master branch likewise before.
> git branch
dev
* master

# change HEAD pointer to dev branch.
> git checkout dev
Switched to branch 'dev'
# do some modification and commit them.

> git commit -am "message"
[master 79635ac] message
  1 file changed, 1 insertion(+)

# switch back to master branch.
> git checkout dev
Switched to branch 'dev'

# open recent changed files. changes does not apply becuz you are working on master branch and commited to dev branch.

# it's possible to create and switch to new branch at once.
> git checkout -b dev master
# master indicates that new branch is started from master.
# -b used to create it.
# to merge branch dev switch to branch you are going to merge with

> git checkout master

# now merge dev. this remove dev branch automatically
# but it keeps revision history (changelog)

> git merge dev
Remotes
# use .gitignore to indicate which files are not going to pushed
# into remote repository. you can put it everywhere in your project
# we use ! to exclude some files and dirs and * for all of them.

# use this command when you want to have a local copy
# from remote repository. --bare is used
> git clone --bare ~/git-repo/

# to clone from github
> git clone git@github.com:smmsadrnezh/repo-name.git

# to see centeral repository url you fetch from or push into
> git remote -v
origin https://github.com/smmsadrnezh/repo-name.git (fetch)
origin https://github.com/smmsadrnezh/repo-name.git (push)
# to push commits
> git push -u origin master

# typing "-u origin master" is only needed at first time.

# to get commits pushed by other collaborators.  
# pull is equivalent to run fetch and merge one by one. 
> git pull

# fetch get's commits but dont merge it.  
# use diff command to see differences.  
> git fetch origin  
> git diff origin/master

# important note: pull everytime you want to push  
# merge conflicts when using three way merge
Rebase
# rebasing branches is not a good idea

> git branch
> gitx
> git rebase master
Thank you :)