# Always Be Delivering

Jimmy Balmert

Robert Cochran

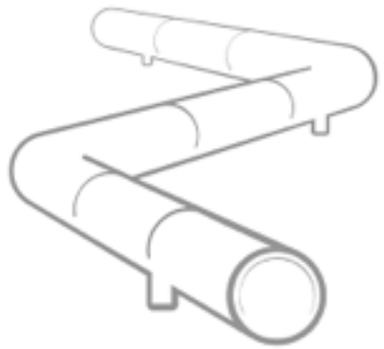
# Agenda

- Introduction
  - Continuous Delivery/Deployment
  - What is a pipeline?
  - Benefits
  - Who Is using CDD?

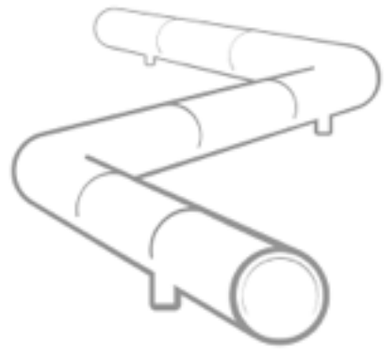
# Agenda (cont.)

- Building a pipeline
  - Environment & Setup
  - Sam automates the build
  - Sam adds versioning to the build
  - Sam pushes artifacts to Nexus
  - Sam adds tests to the build
  - Sam adds health checks to the app
  - Sam adds acceptance tests to the build
  - An aside about QA
  - Sam adds parallel tests to the build
  - Sam adds feature toggles to the app
  - Sam uses feature toggles for a/b testing
  - Sam automates deployments to production
  - Sam adds blue-green deployments
  - Add a job to poll SCM
  - Sam adds code coverage
  - Sam adds a dashboard

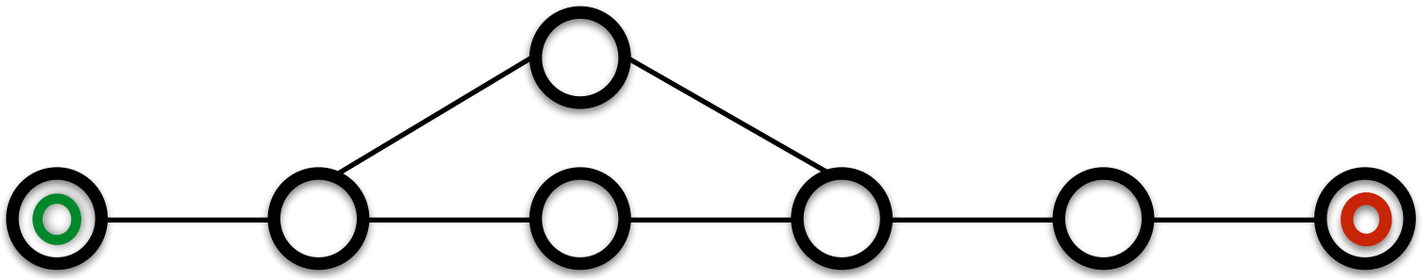
## Continuous Delivery



## Continuous Deployment

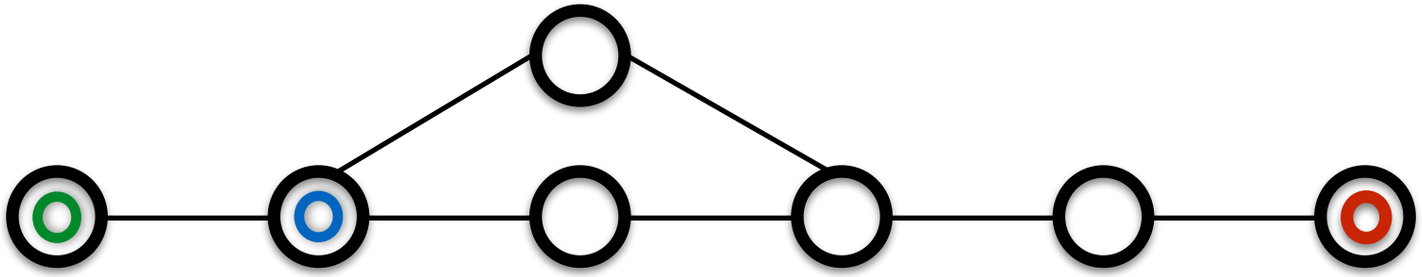


# What is a pipeline?

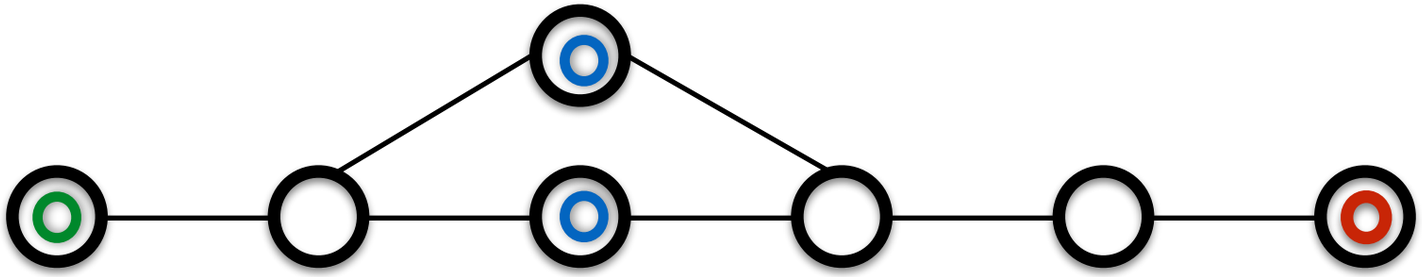


# What is a pipeline?

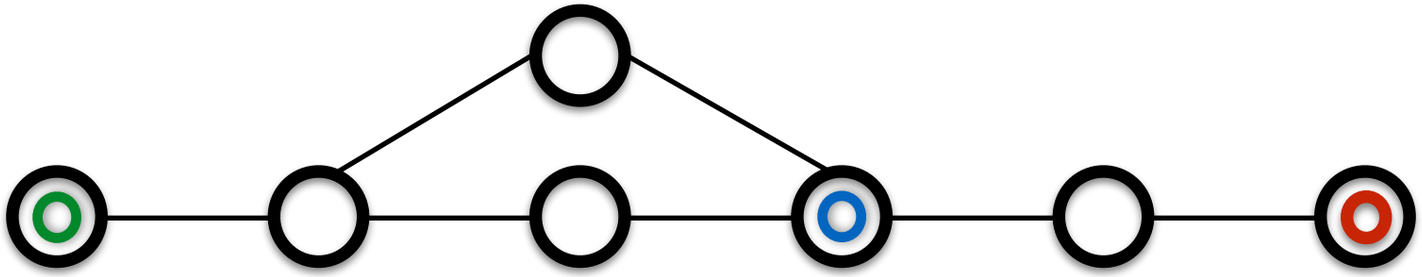
# What is a pipeline?



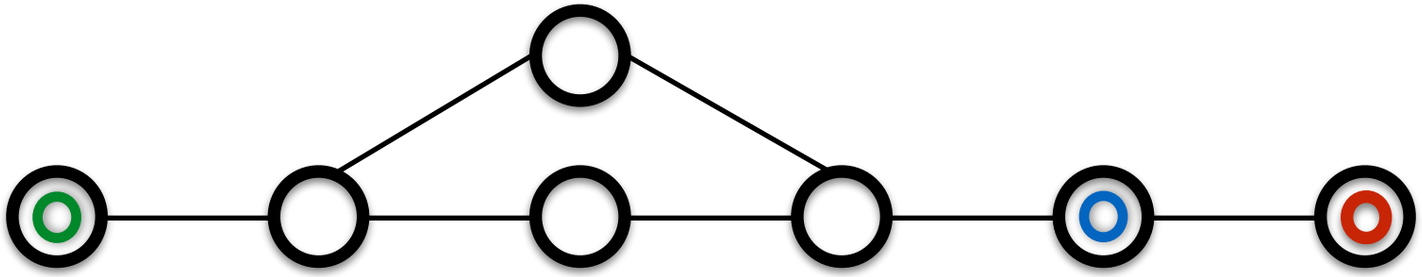
# What is a pipeline?



# What is a pipeline?



# What is a pipeline?



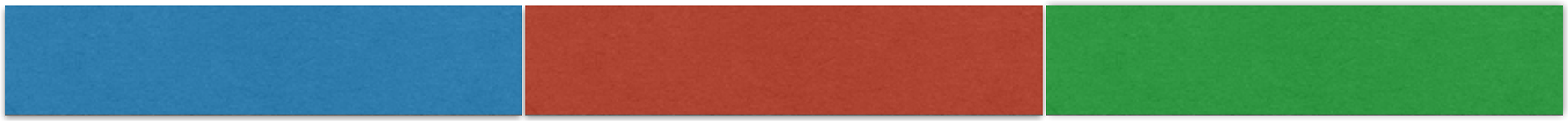
# What is a pipeline?

# Benefits

# Standard development timeline

Development complete

Feature deployed to prod

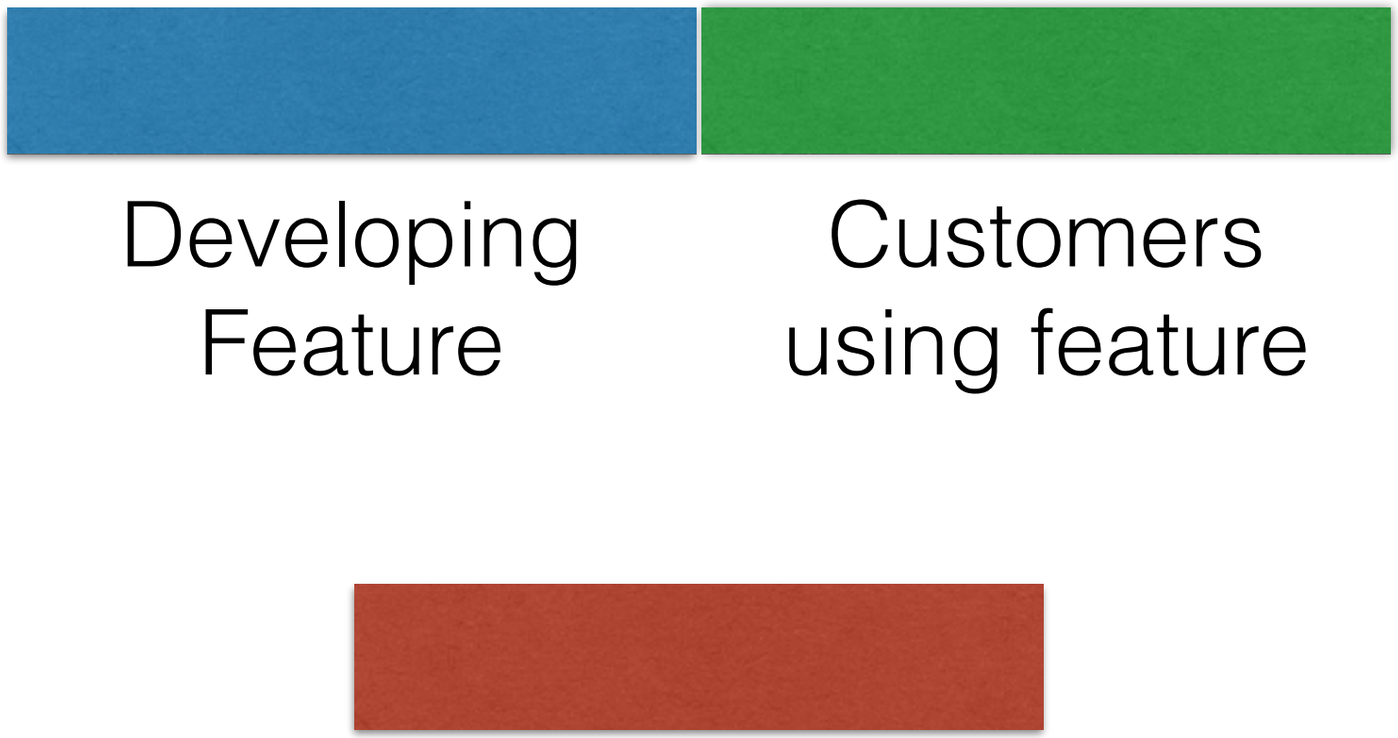


Developing Feature

Waiting

Customers using feature

# Continuous Delivery timeline

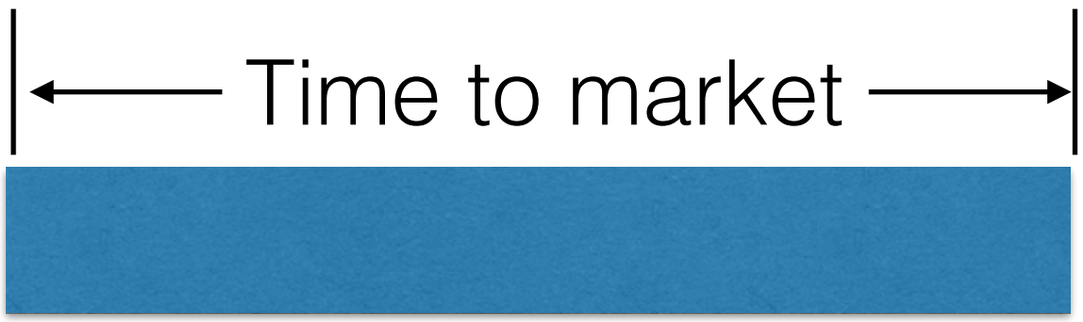


Developing Feature

Customers using feature

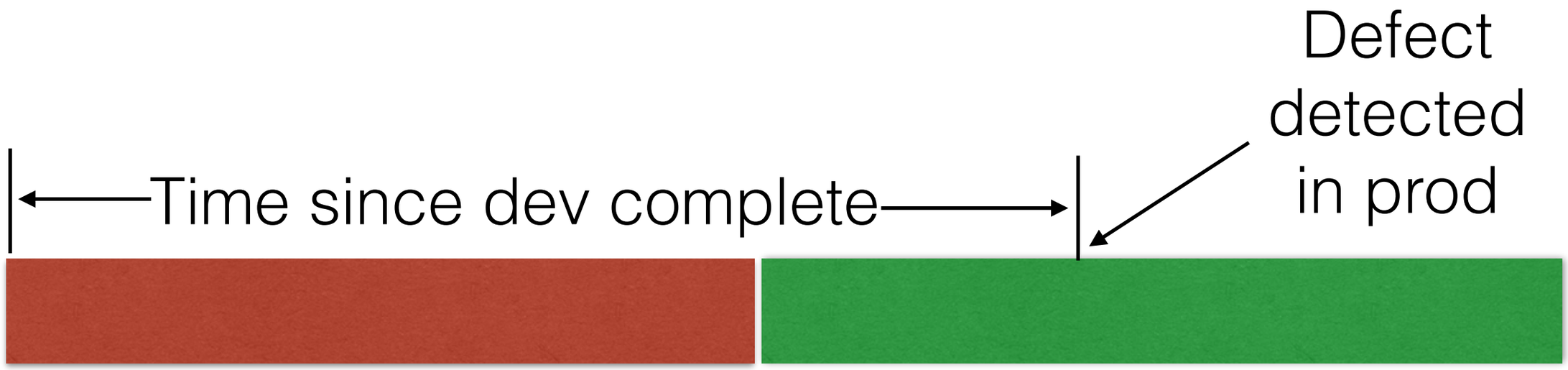
# Standard timeline


# Continous Delivery timeline

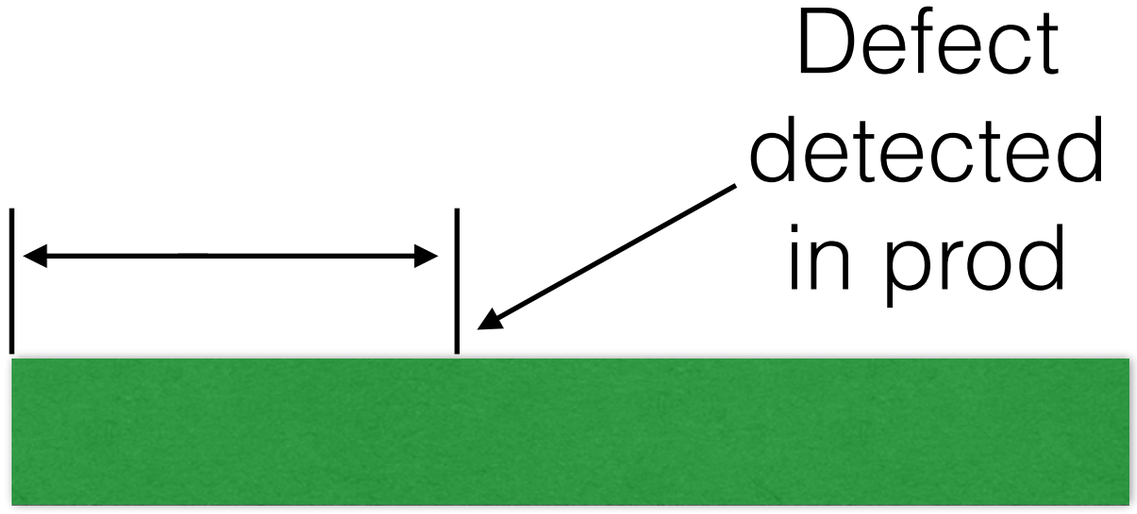


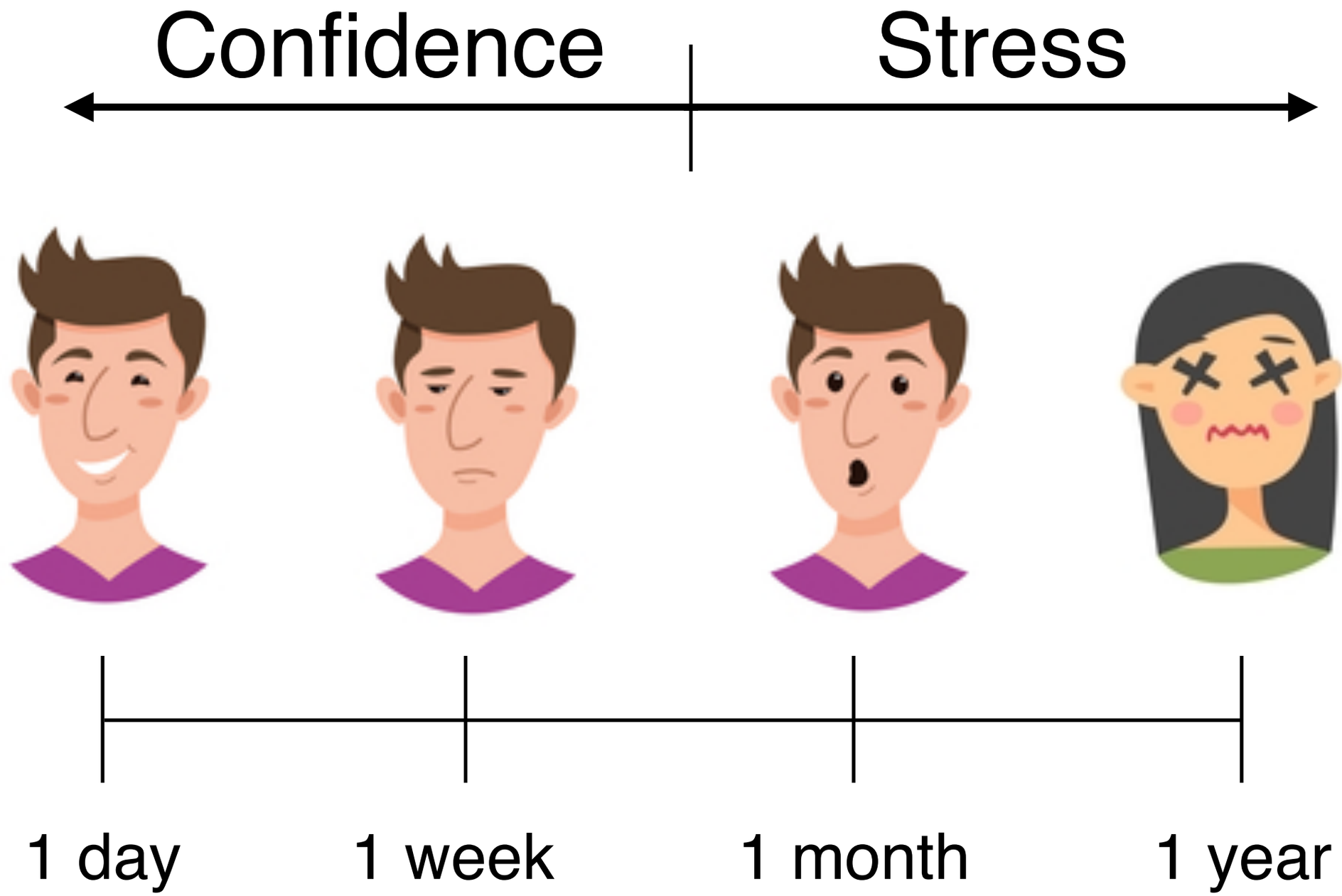
# Get to market faster

# Standard timeline




# Continous Delivery timeline

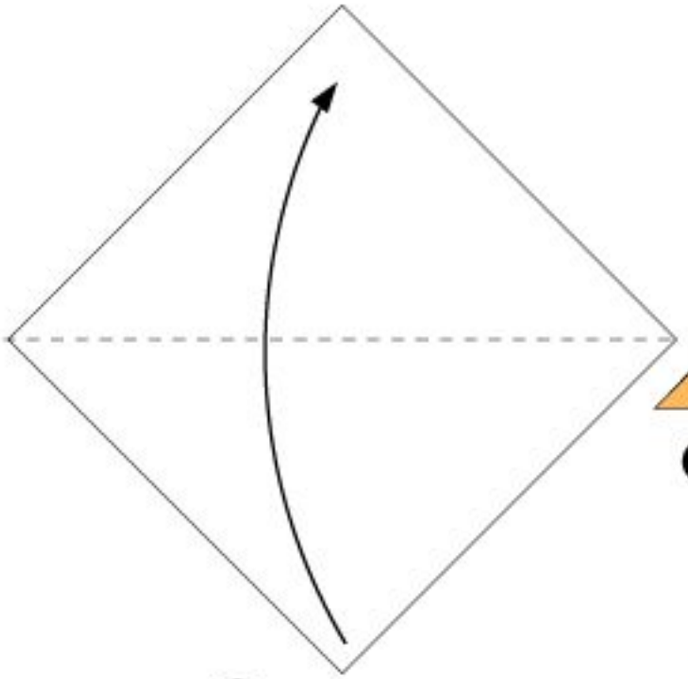




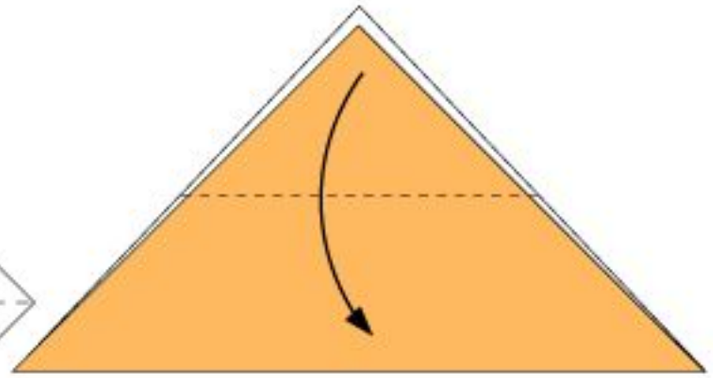

Time since you worked on a feature

Greater confidence

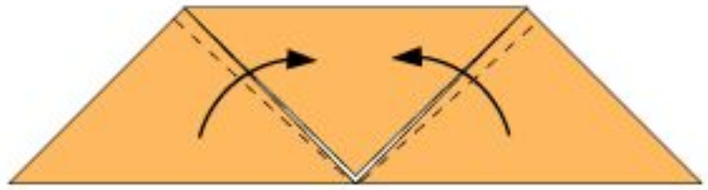
Lower stress



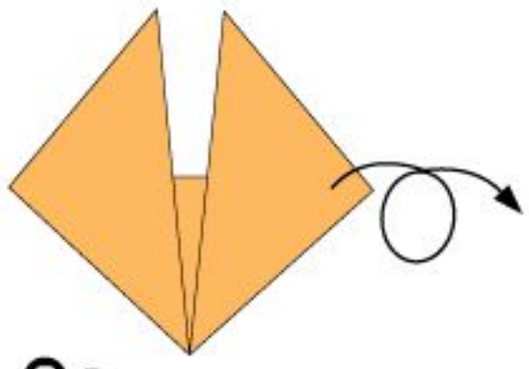
❶ Fold in half



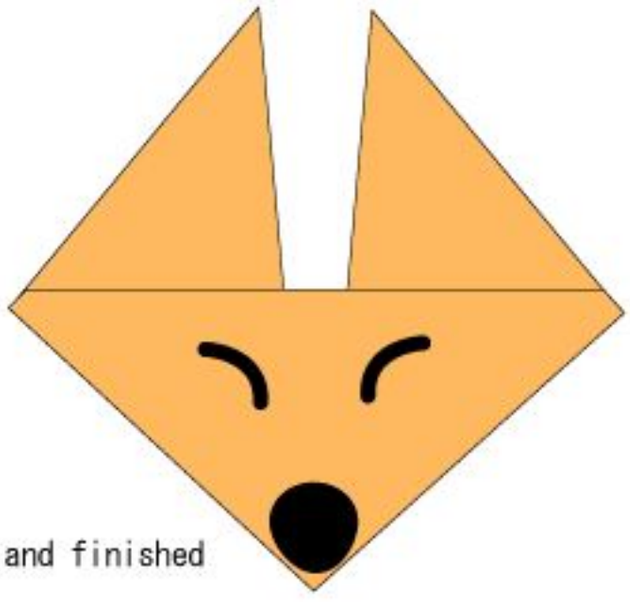
❷ Fold in the dotted line



❹ Fold in the dotted line



❸ Turn over



❺ Draw eyes, a nose and finished

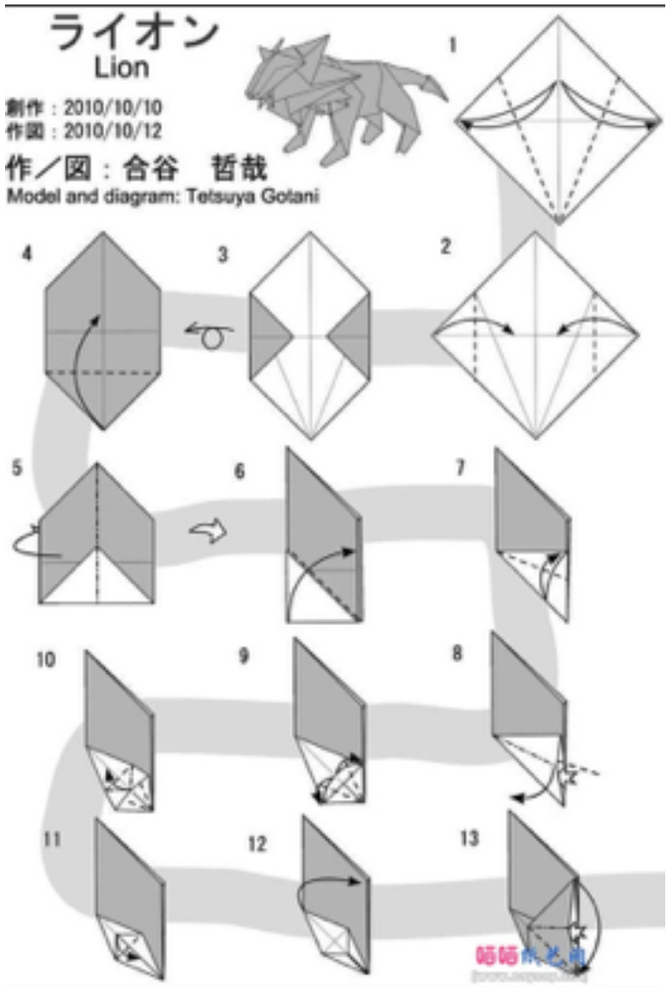
# A Fox (face)

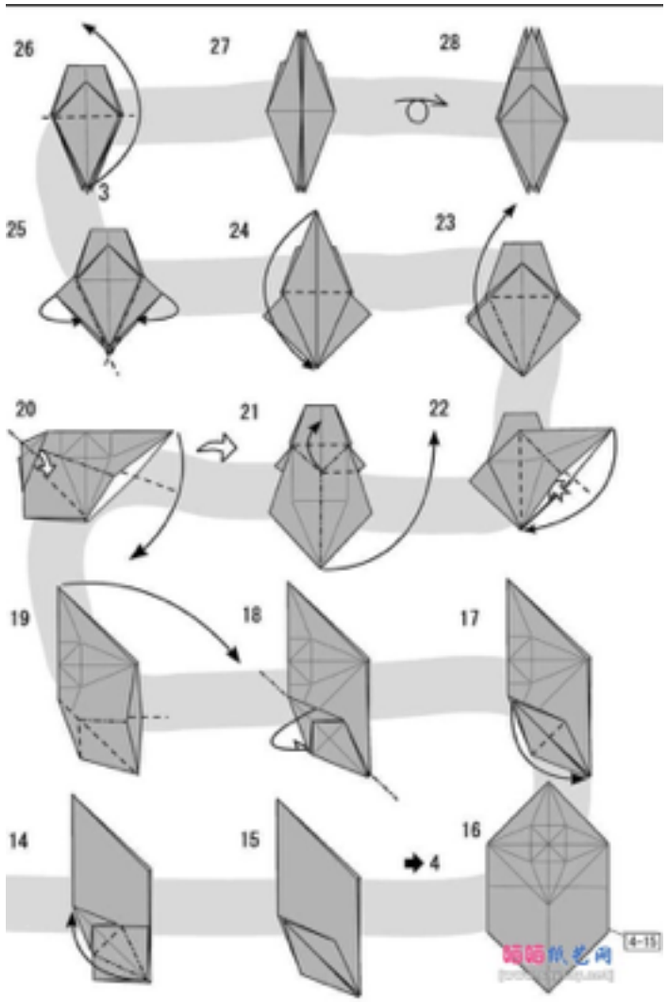
*Traditional
Diagram:Fumiaki Shingu

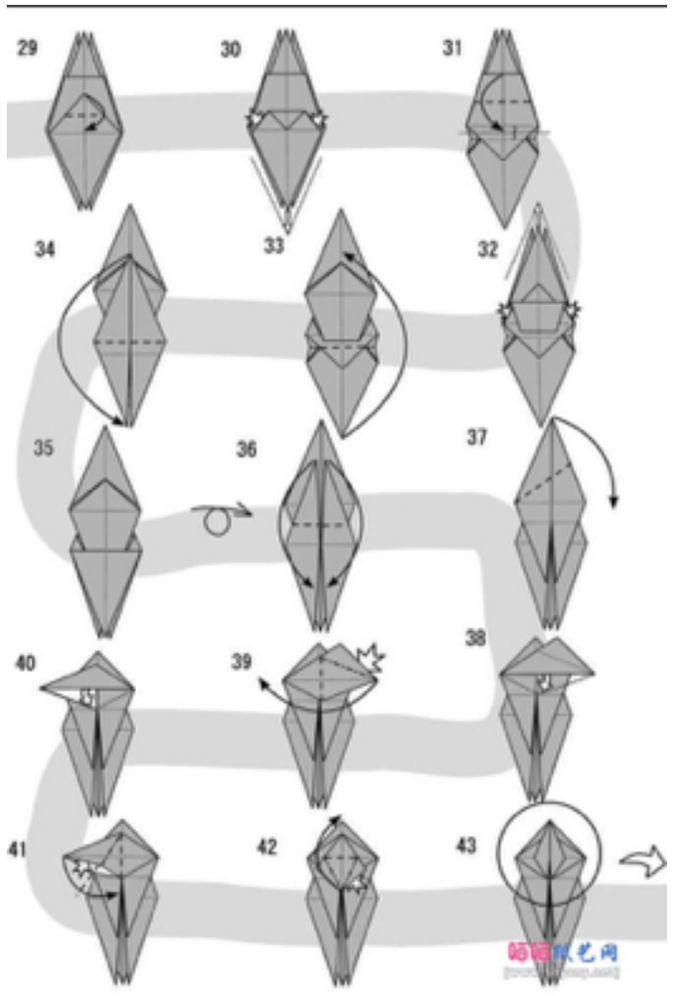
# ライオン
## Lion

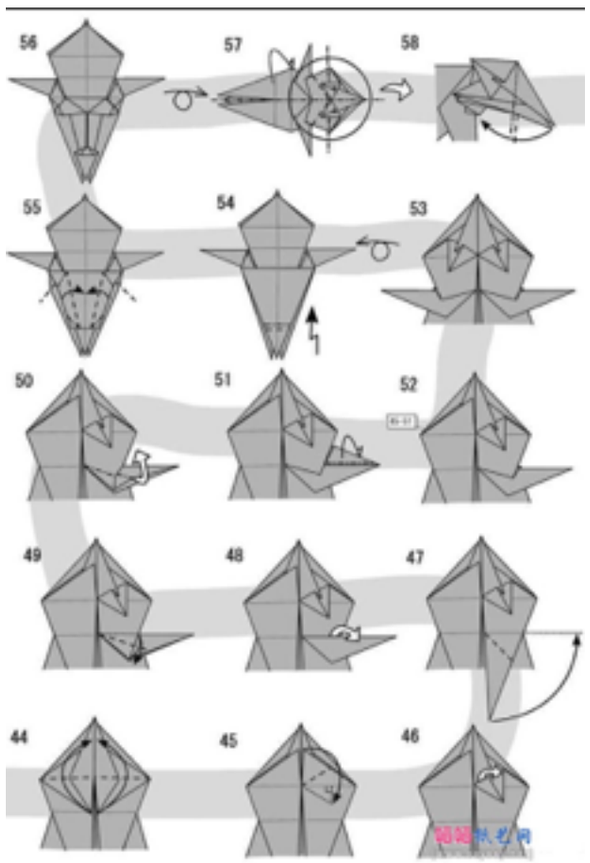
創作：2010/10/10
作図：2010/10/12

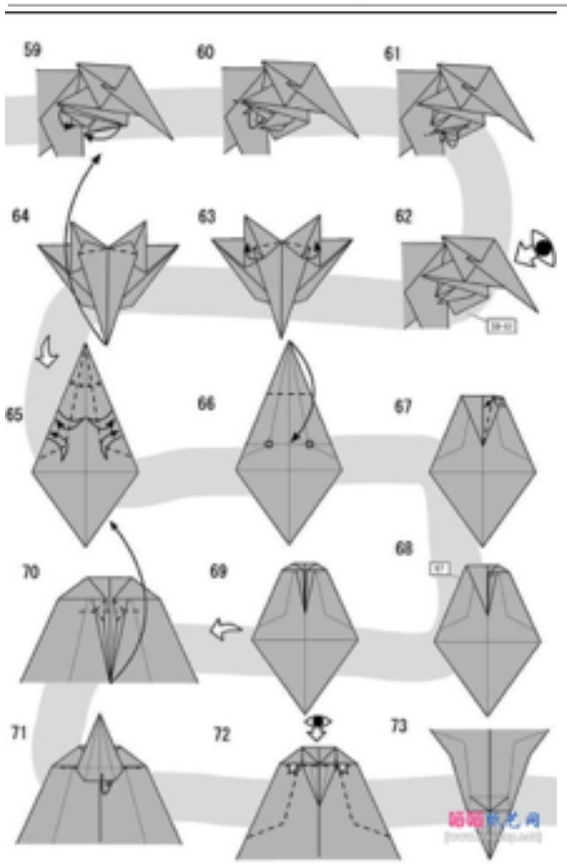
作／図：合谷　哲哉
Model and diagram: Tetsuya Gotani

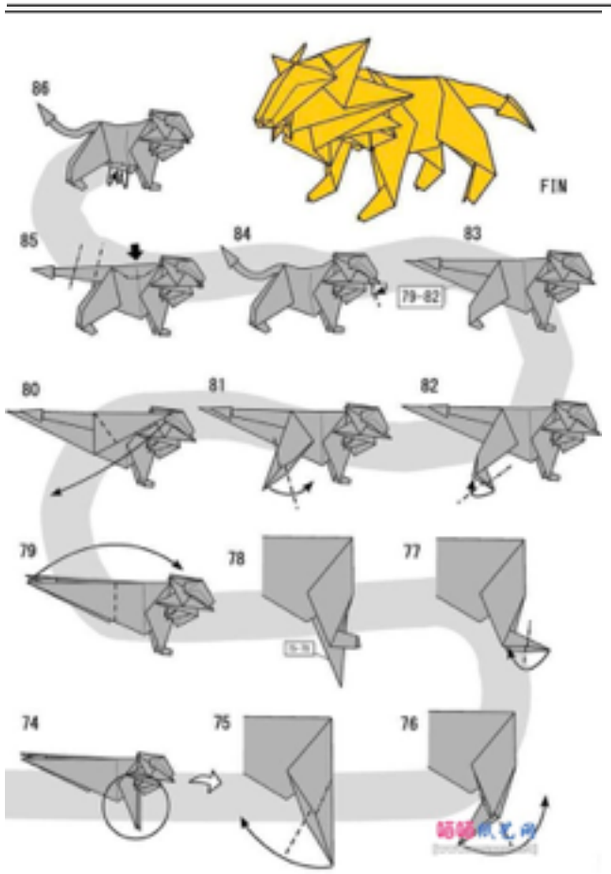


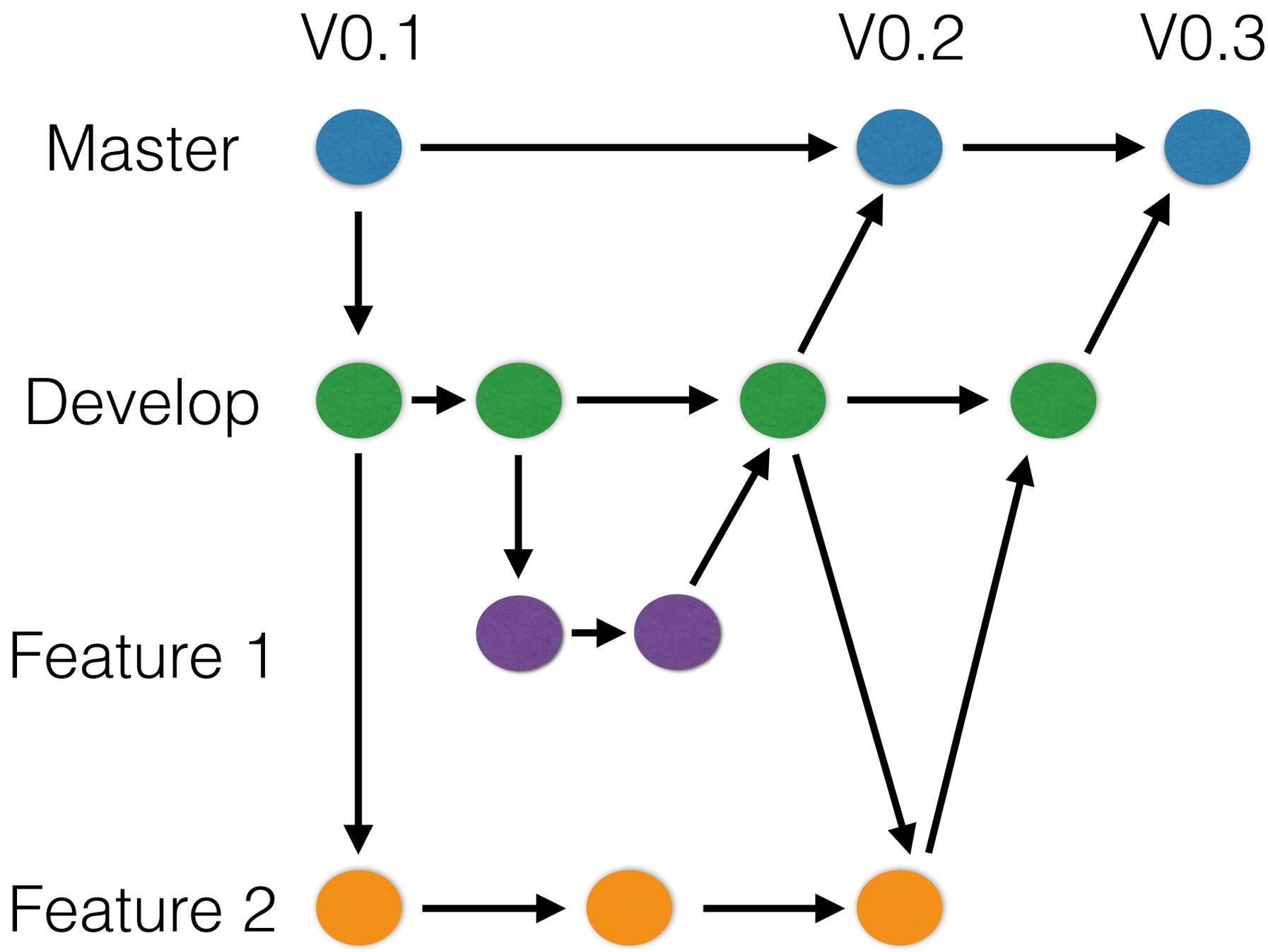



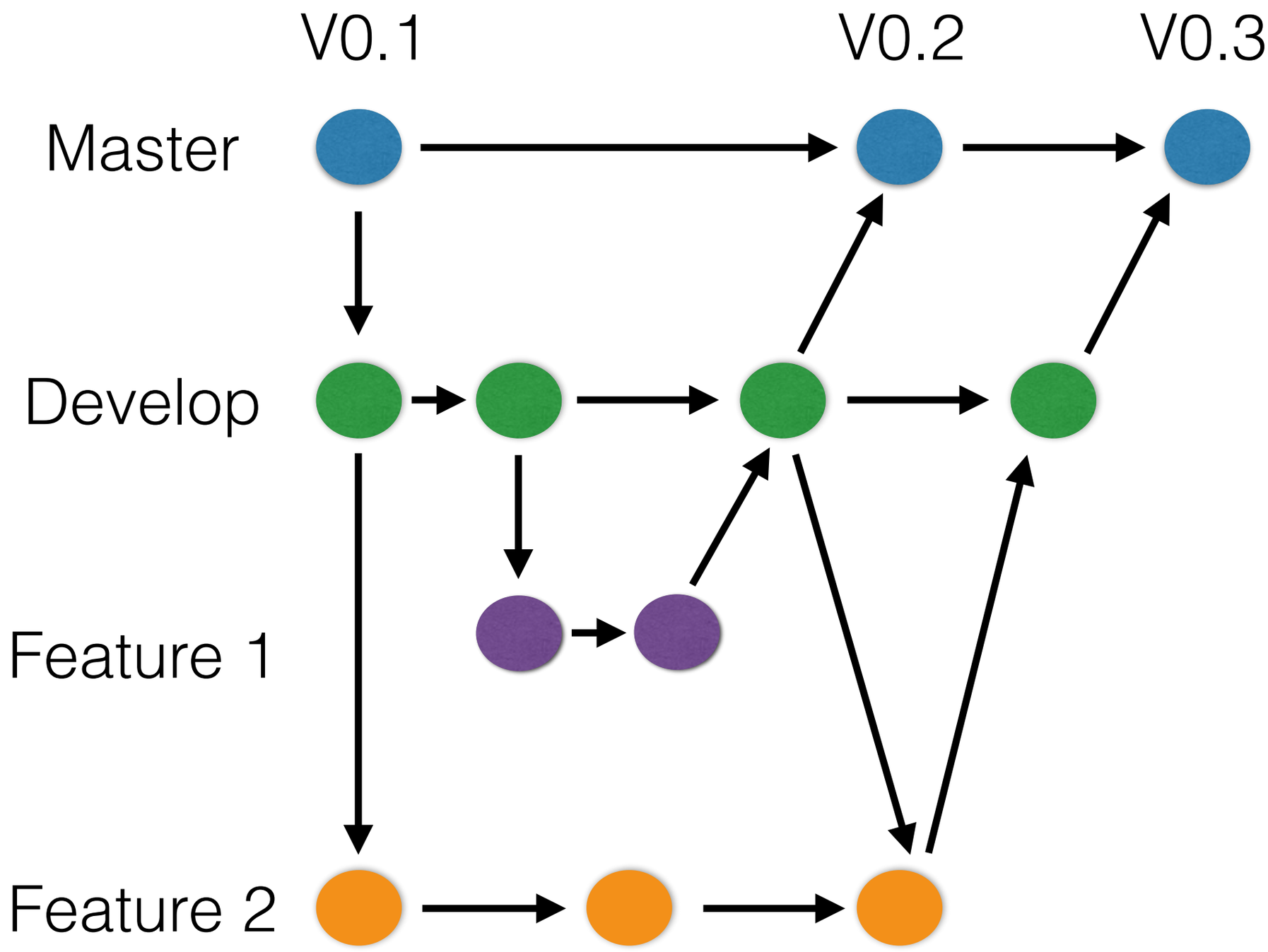
# Smaller changes imply less risk

# Master = Prod

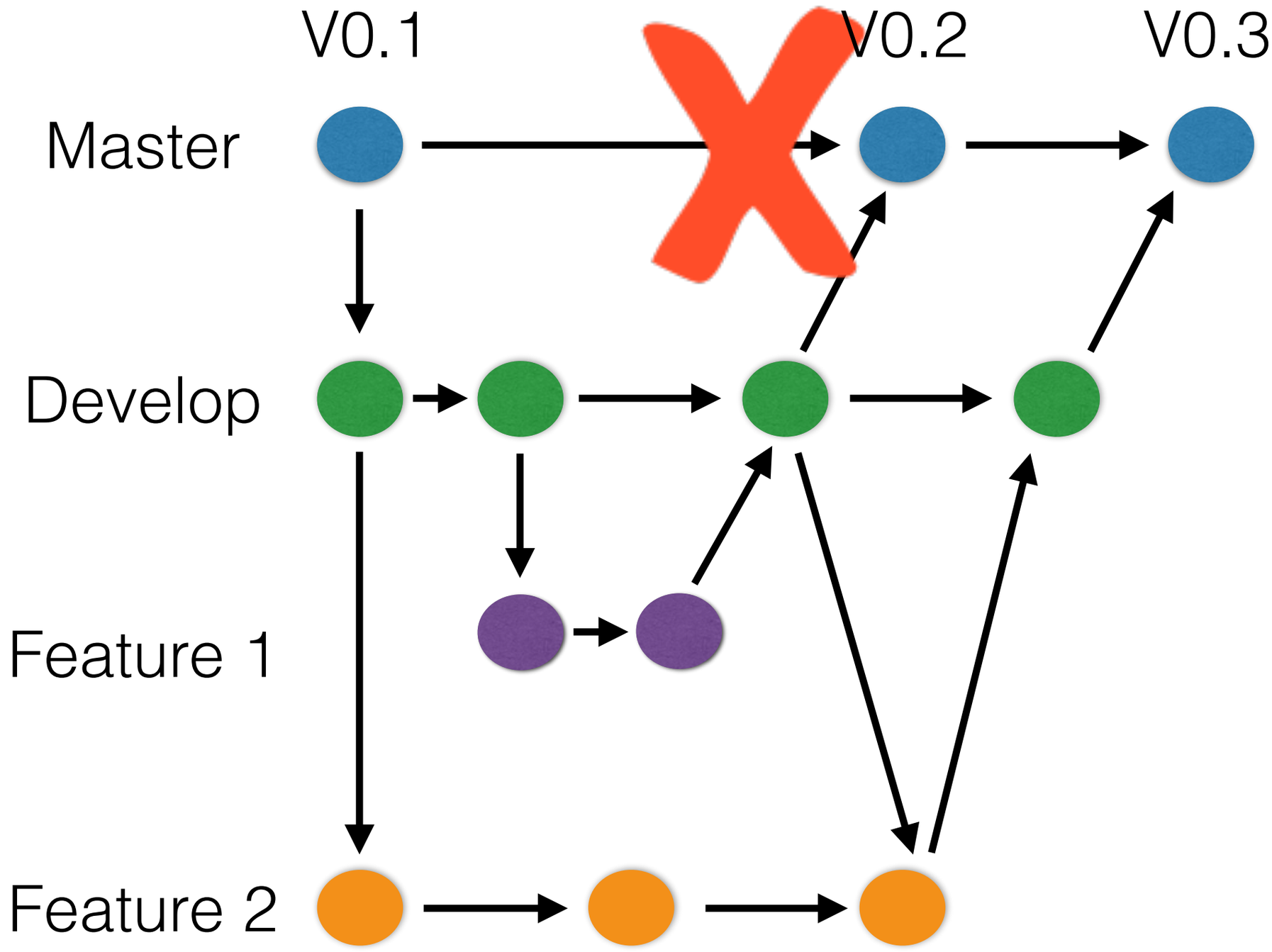


# Master = Latest

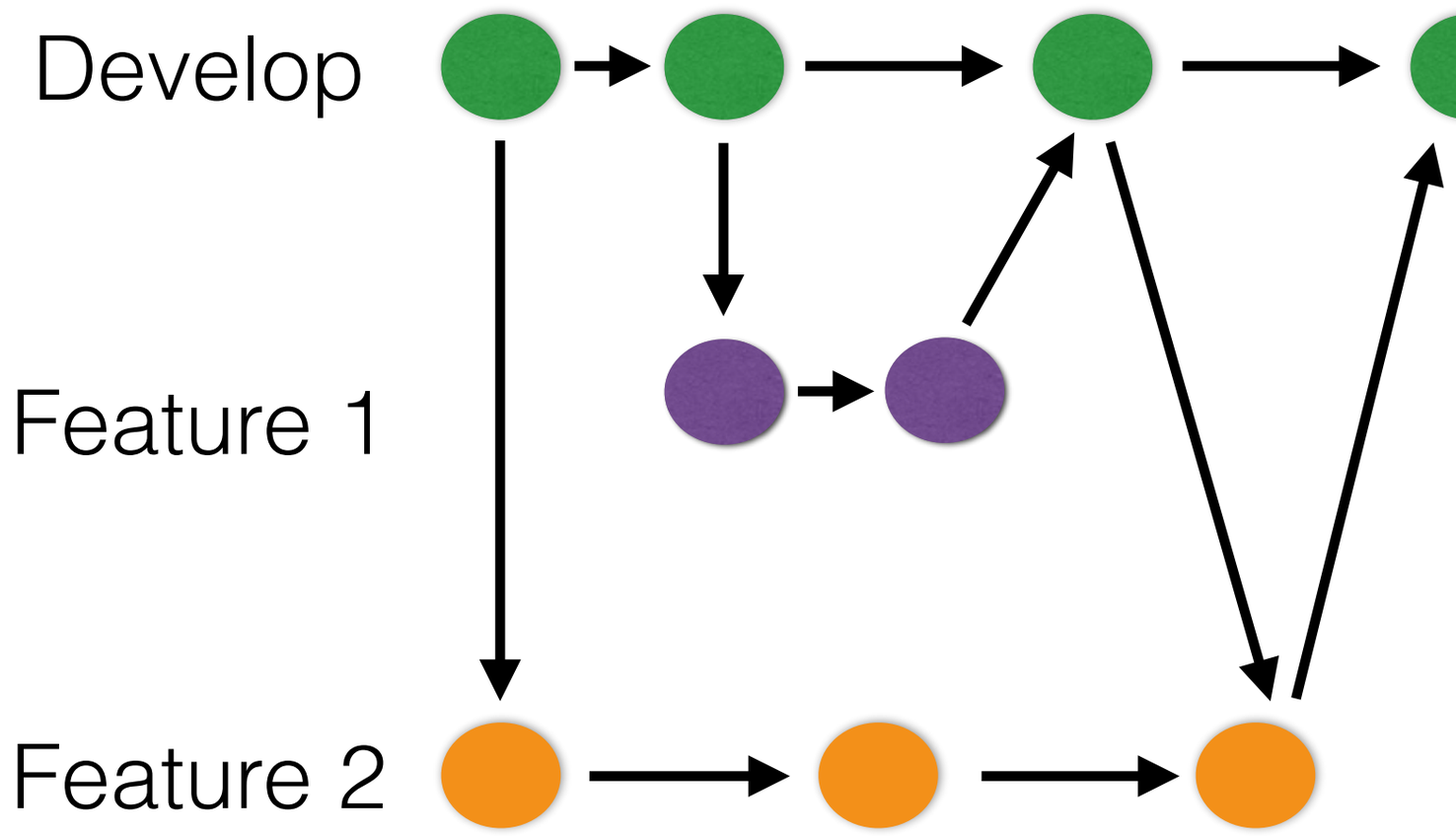
# Master = Latest



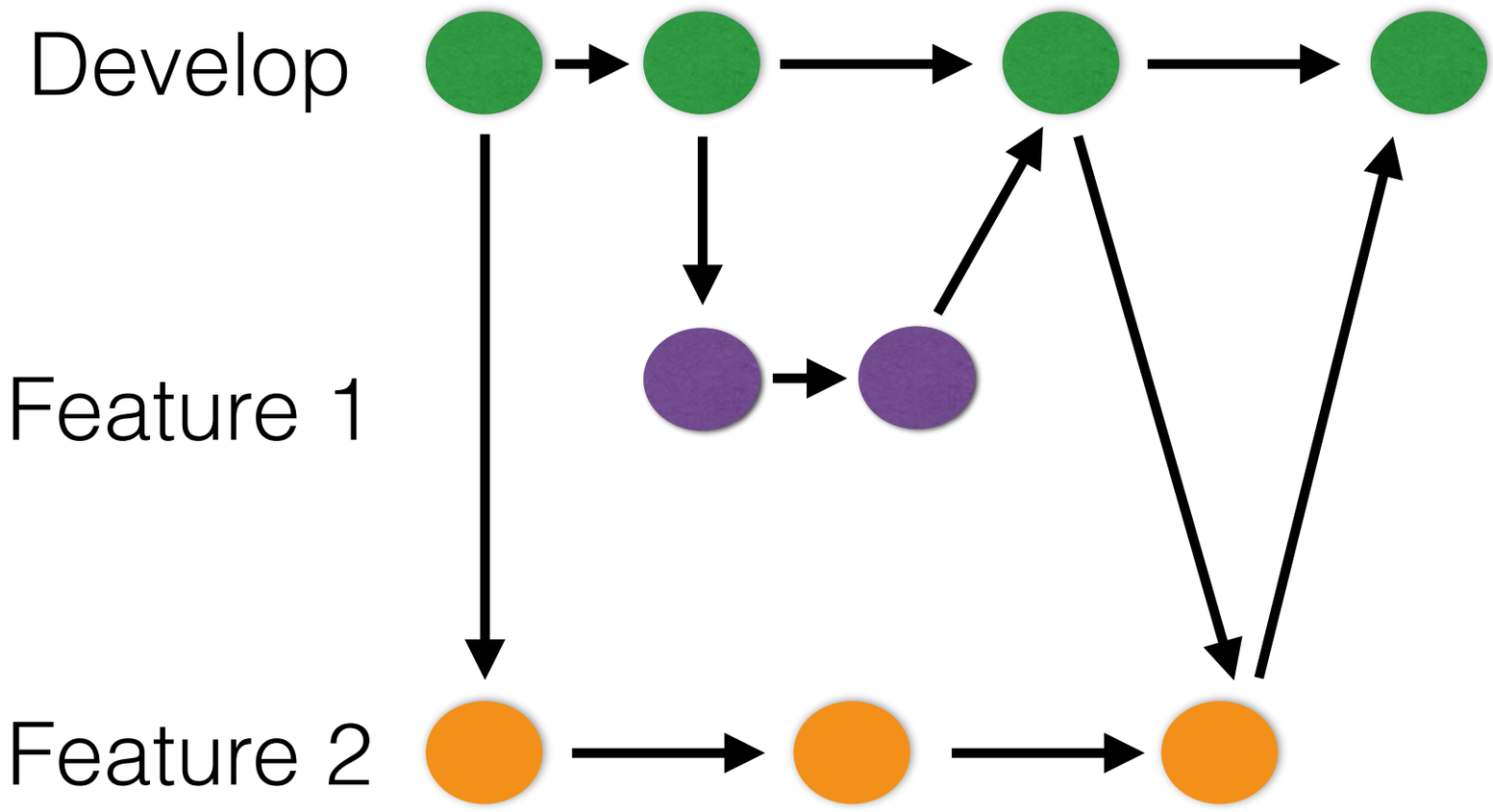
# Master = Latest



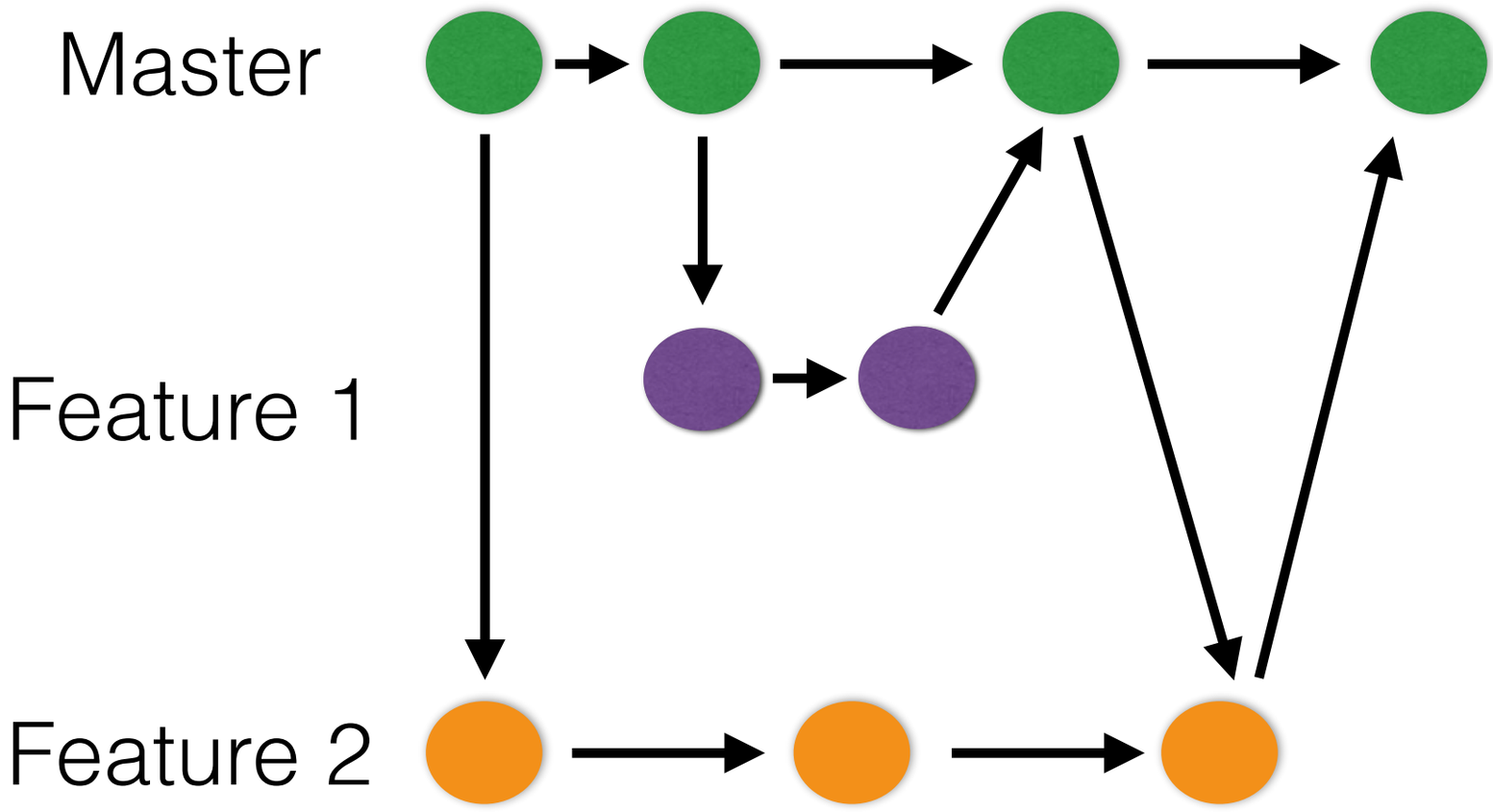
# Master = Latest



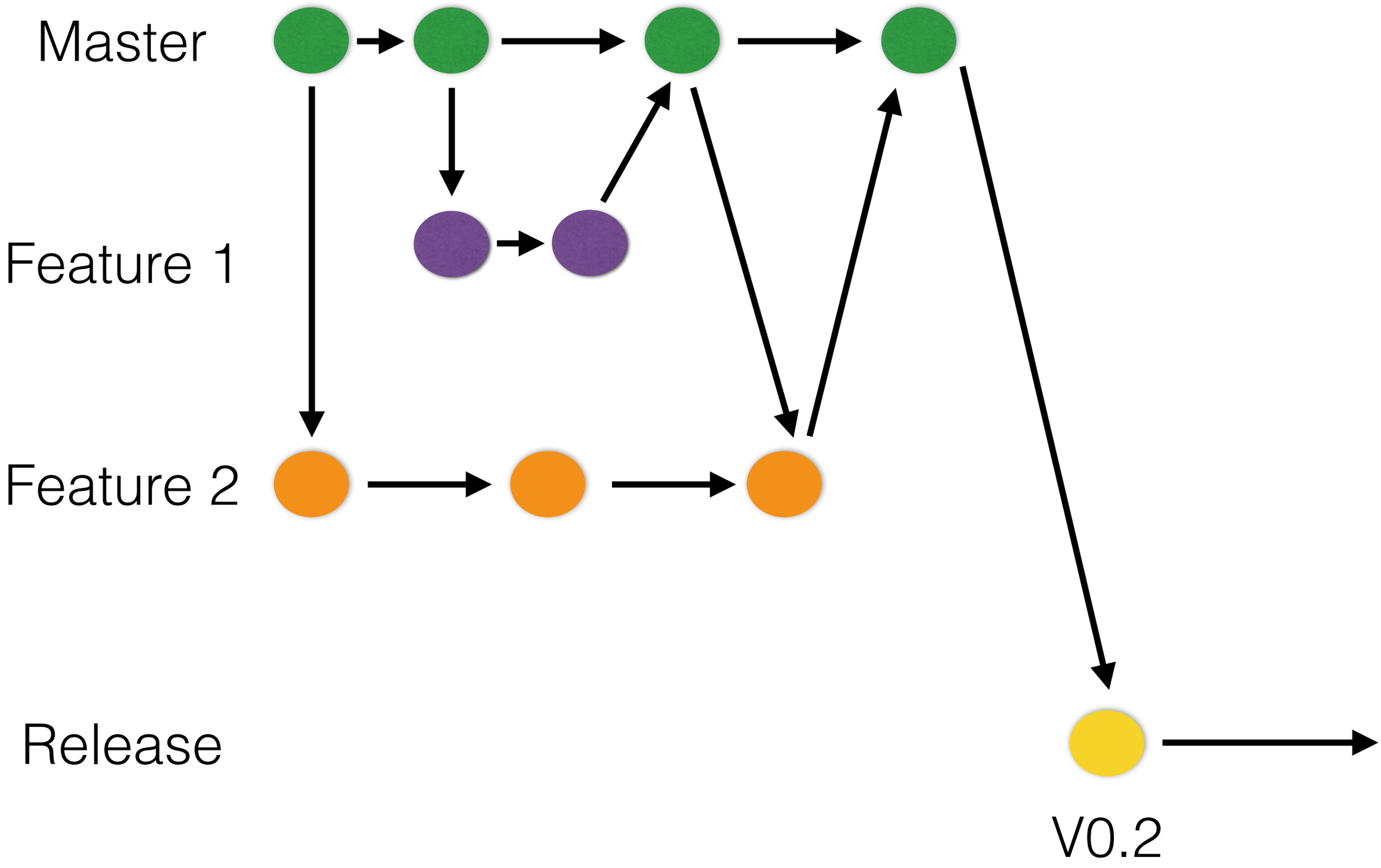
# Master = Latest



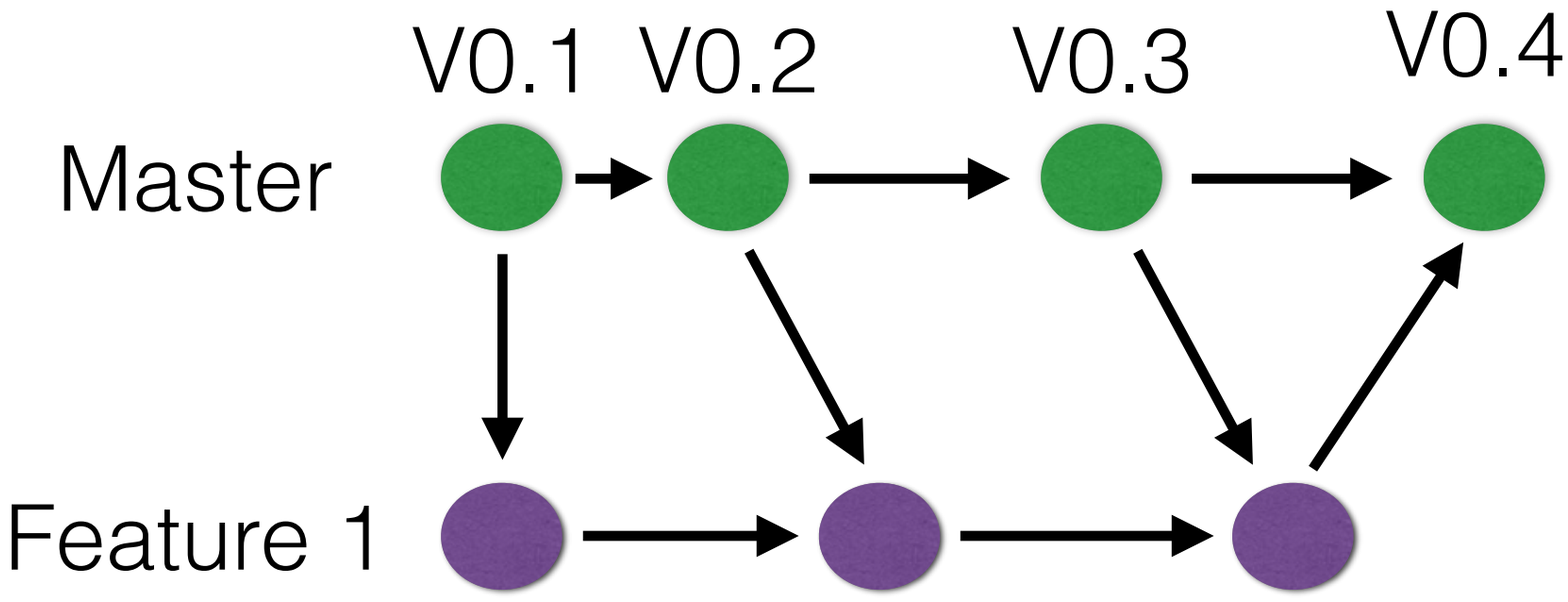
# Master = Latest



# Master = Latest



# Master = Continuous Delivery

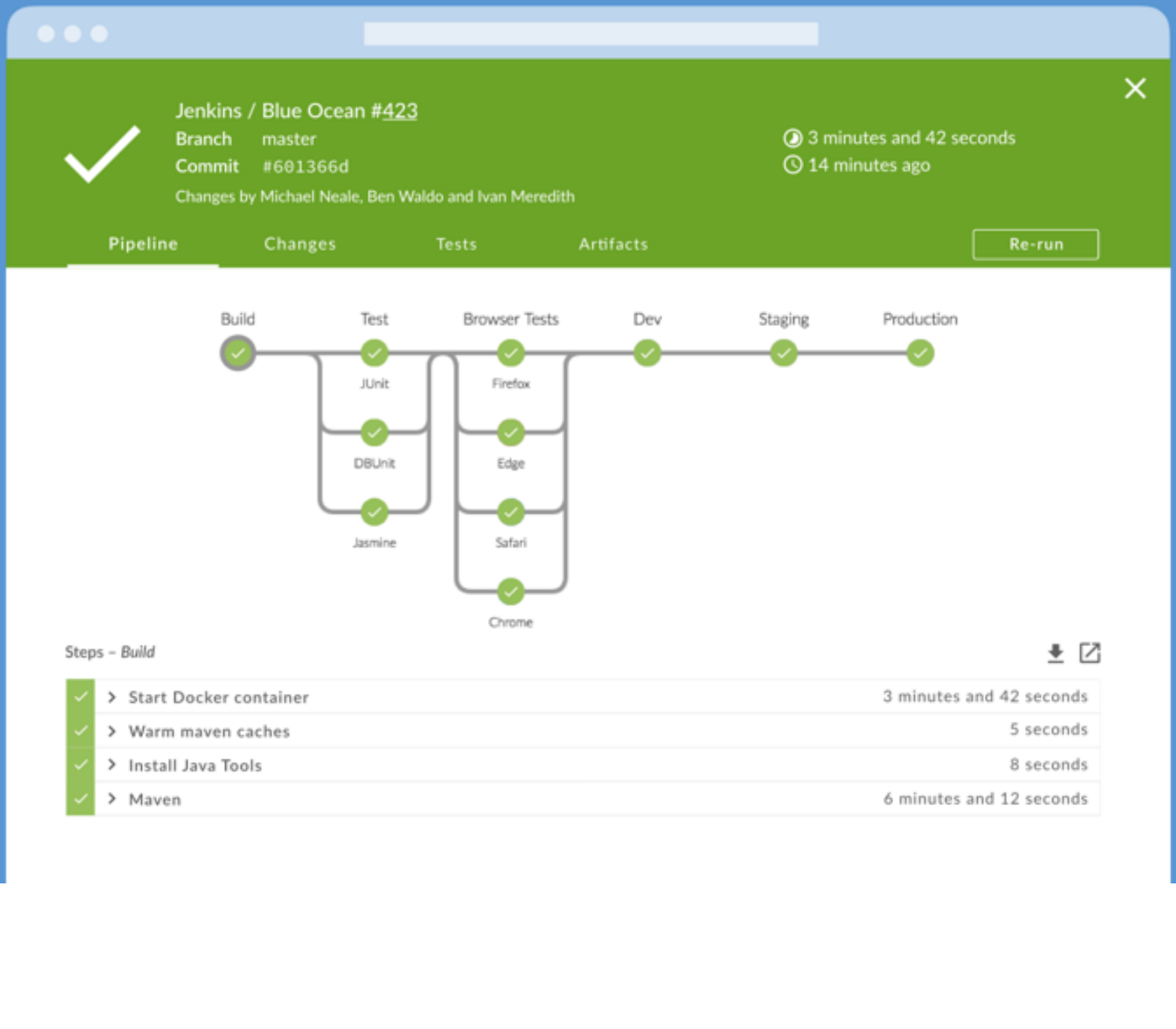


# Simpler branch strategy

But, why a pipeline?

# Visibility

| | build | test: integration-&-quality | test: functional | test: load-&-security | approval | deploy: prod |
|---|---|---|---|---|---|---|
| Average stage times: (Average full run time: ~5s) | 836ms | 20min 43s | 9ms | 7ms | 89ms | 5ms |
| #17 Sep 22 15:05 — No Changes — Retry — Download | 538ms master | 10s master | 10ms master | 8ms master | 72ms (paused for 7s) master | 4ms master |
| #16 Sep 22 15:04 — No Changes — Retry — Download | 479ms master | 6s master | 9ms master | 9ms master | 74ms (paused for 6s) master | 5ms master |
| #15 Sep 22 15:03 — No Changes — Retry — Download | 922ms master | 6s master | 10ms master | 9ms master failed | | |
| #14 Sep 22 15:03 — No Changes — Retry — Download | 1s master | 8s master | 12ms master | 9ms master | 80ms (paused for 5s) master | 5ms master |
| #13 Sep 22 15:02 — No Changes — Download | 942ms master | 9s master | 13ms master failed | | | |
| #12 Sep 22 15:02 — No Changes — Retry — Download | 1s master | 6s master | 13ms master | 11ms master | 111ms (paused for 5s) master aborted | |


Jenkins / Blue Ocean #423
Branch master
Commit #601366d
Changes by Michael Neale, Ben Waldo and Ivan Meredith
3 minutes and 42 seconds
14 minutes ago
Pipeline
Changes
Tests
Artifacts
Re-run
Build
Test
Browser Tests
Dev
Staging
Production
JUnit
DBUnit
Jasmine
Firefox
Edge
Safari
Chrome
Steps – Build
Start Docker container
3 minutes and 42 seconds
Warm maven caches
5 seconds
Install Java Tools
8 seconds
Maven
6 minutes and 12 seconds

# Visibility

Google

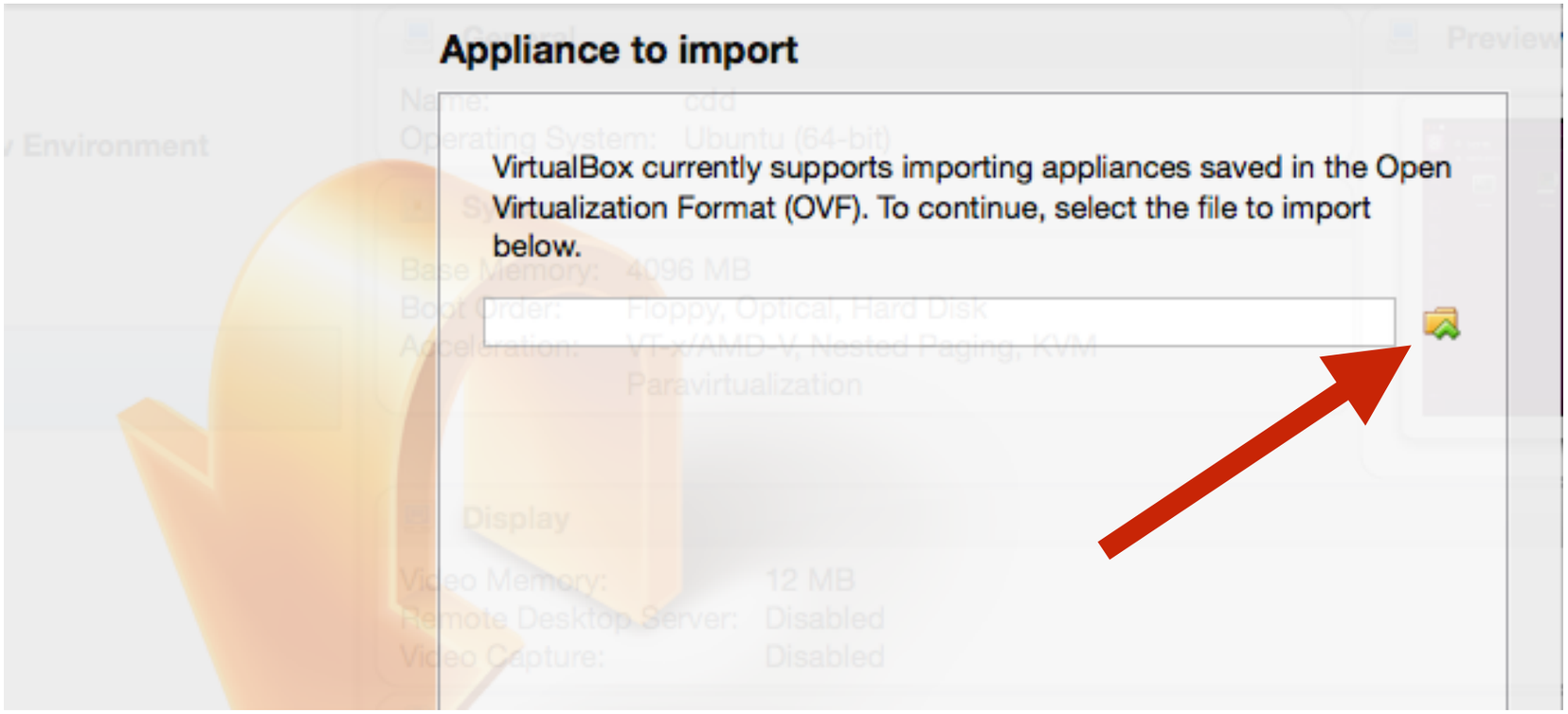
TESLA

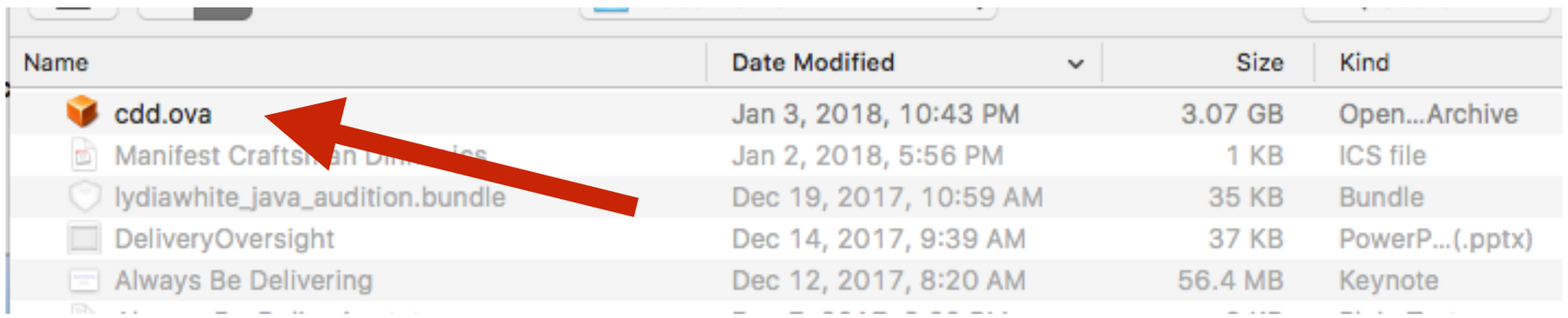
NETFLIX

PDF

# Lab 0:
# Environment & Setup

- App directory structure
- Checkout code from local origin
- Login into Jenkins
  - Add credentials for Nexus
  - Add settings.xml file for Maven
- Login to Nexus


Appliance to import
VirtualBox currently supports importing appliances saved in the Open Virtualization Format (OVF). To continue, select the file to import below.


Name
Date Modified
Size
Kind
cdd.ova
Jan 3, 2018, 10:43 PM
3.07 GB
Open...Archive
Jan 2, 2018, 5:56 PM
1 KB
ICS file
lydiawhite_java_audition.bundle
Dec 19, 2017, 10:59 AM
35 KB
Bundle
DeliveryOversight
Dec 14, 2017, 9:39 AM
37 KB
PowerP...(.pptx)
Always Be Delivering
Dec 12, 2017, 8:20 AM
56.4 MB
Keynote

cdd - Network

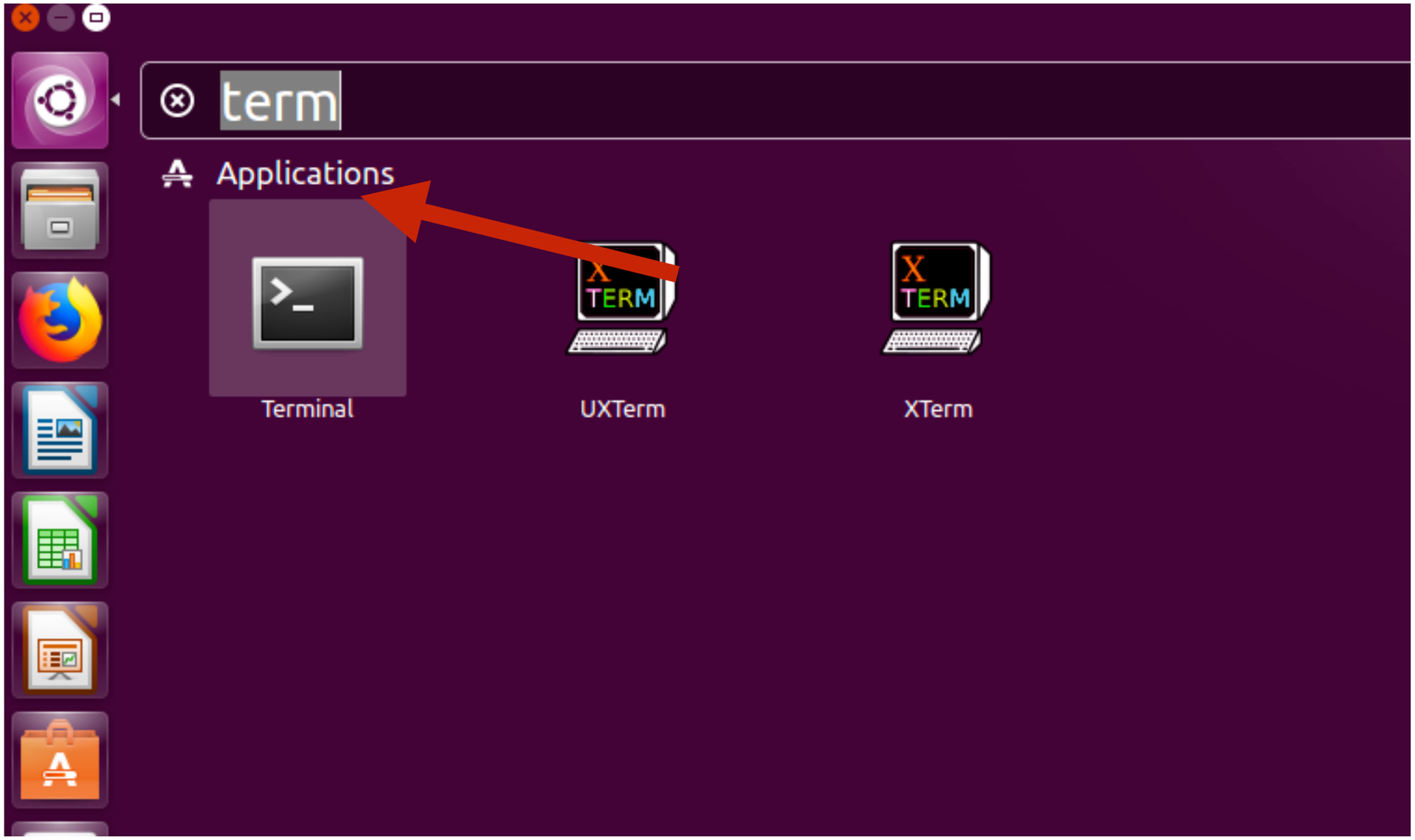
General | System | Display | Storage | Audio | Network | Ports | Shared Folders | User Interface

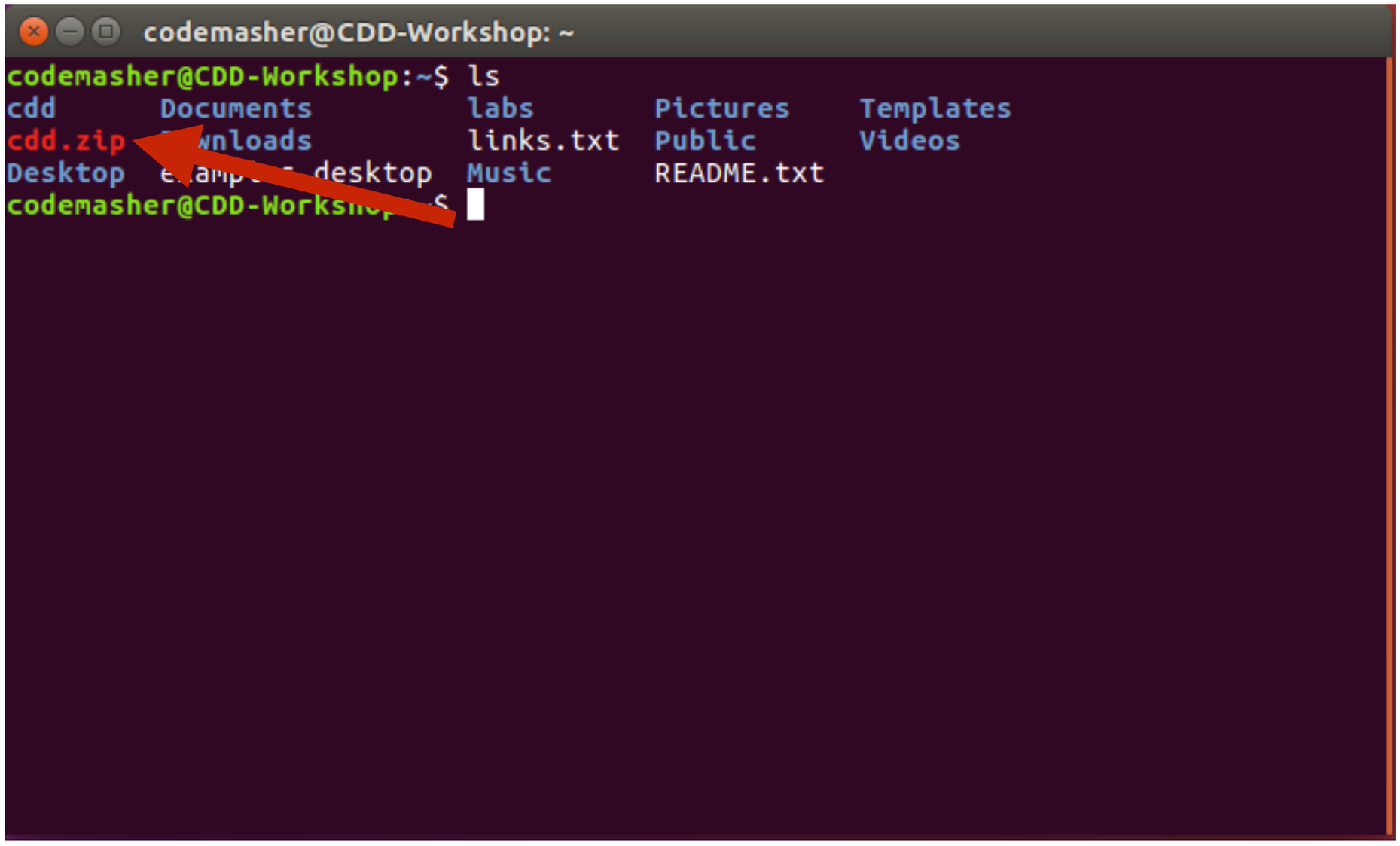
| Name | Protocol | Host IP | Host Port | Guest IP | Guest Port |
|---|---|---|---|---|---|
| Rule 1 | TCP | 127.0.0.1 | 9418 | 10.0.2.15 | 9418 |
| Rule 2 | TCP | 127.0.0.1 | 9080 | 10.0.2.15 | 9080 |
| Rule 3 | TCP | 127.0.0.1 | 9081 | 10.0.2.15 | 9081 |
| Rule 4 | TCP | 127.0.0.1 | 8091 | 10.0.2.15 | 8091 |
| Rule 5 | TCP | 127.0.0.1 | 10001 | 10.0.2.15 | 10002 |
| Rule 6 | TCP | 127.0.0.1 | 10002 | 10.0.2.15 | 10002 |

Cancel OK

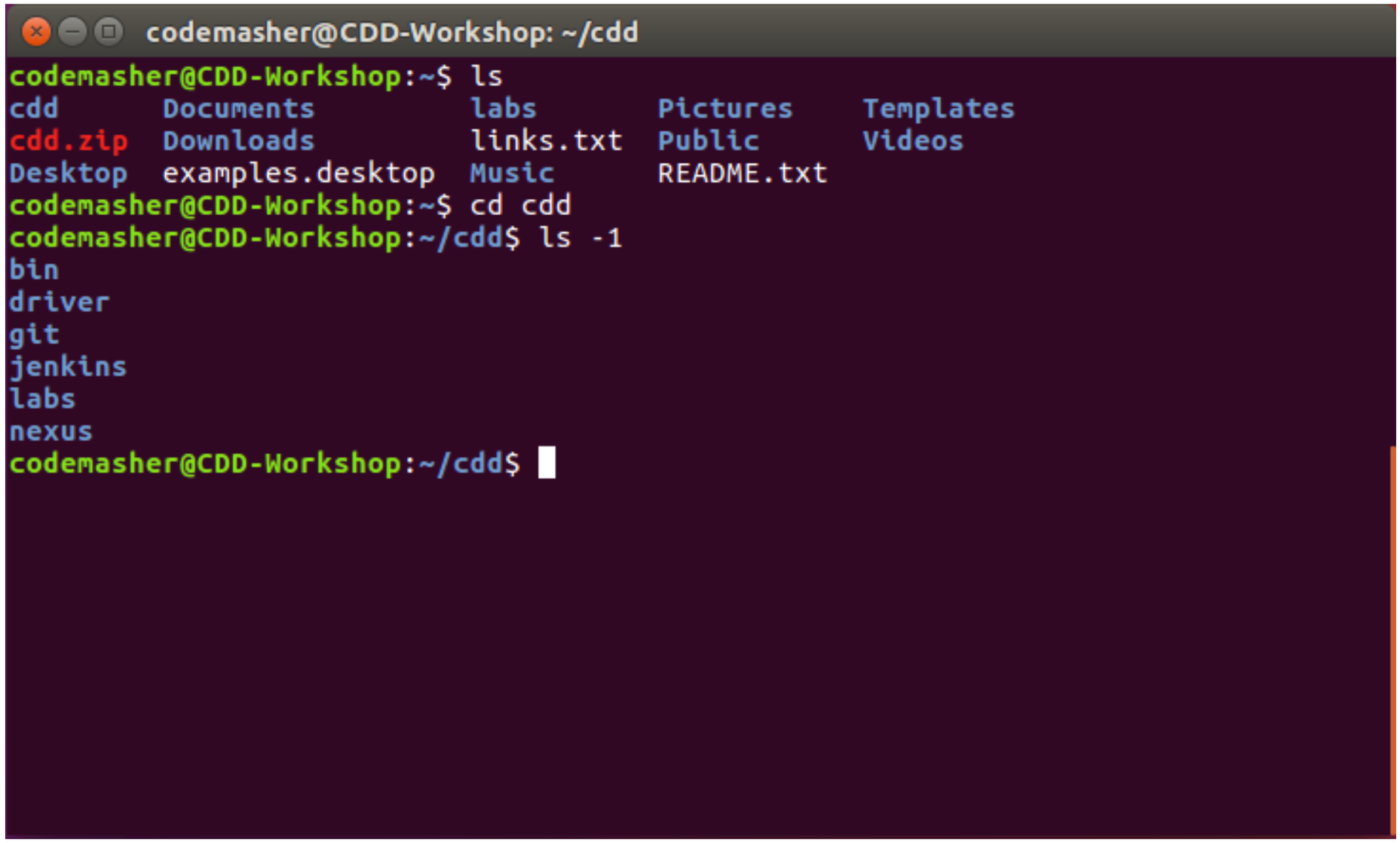
?

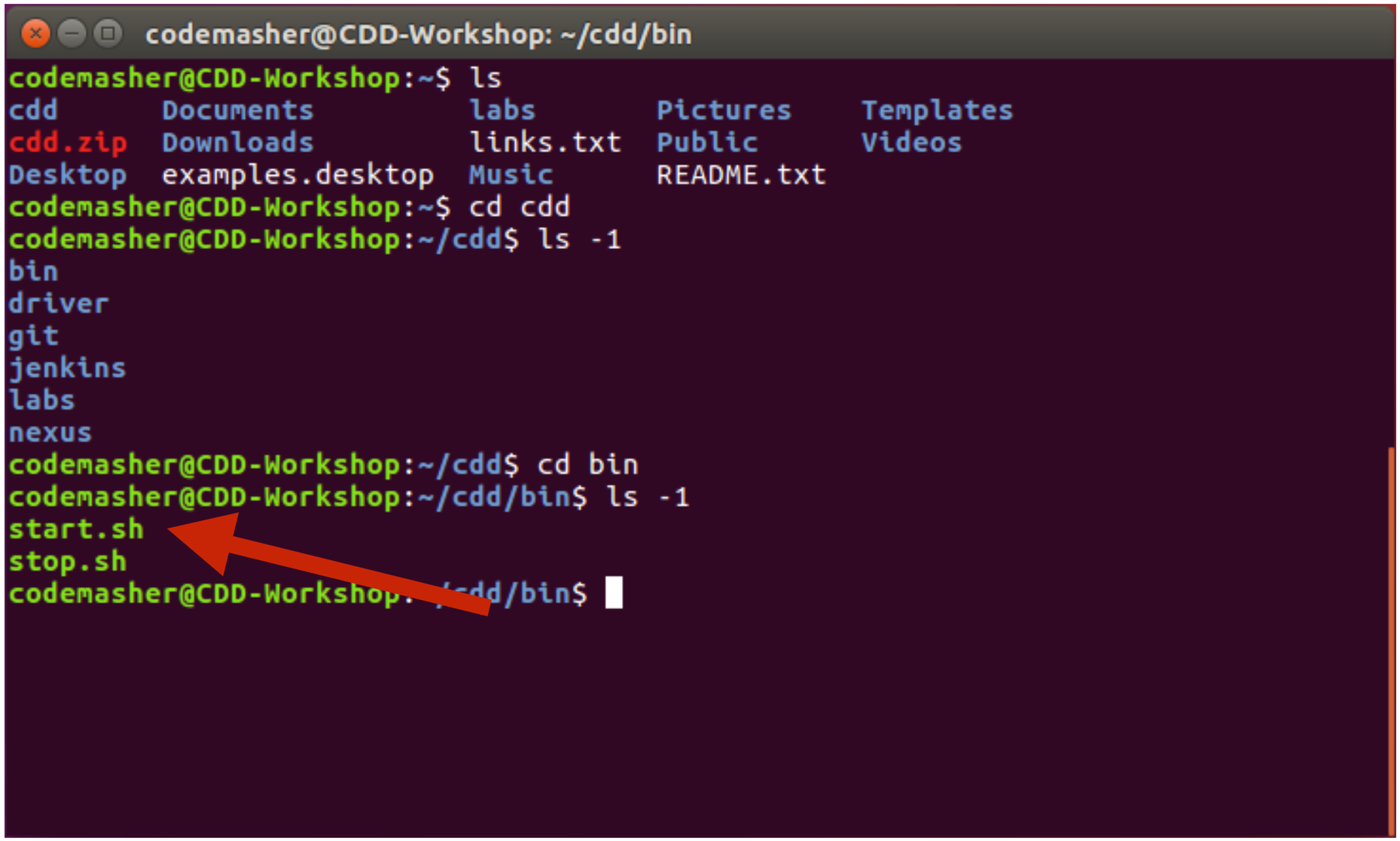
Cancel OK


term
Applications
Terminal
UXTerm
XTerm

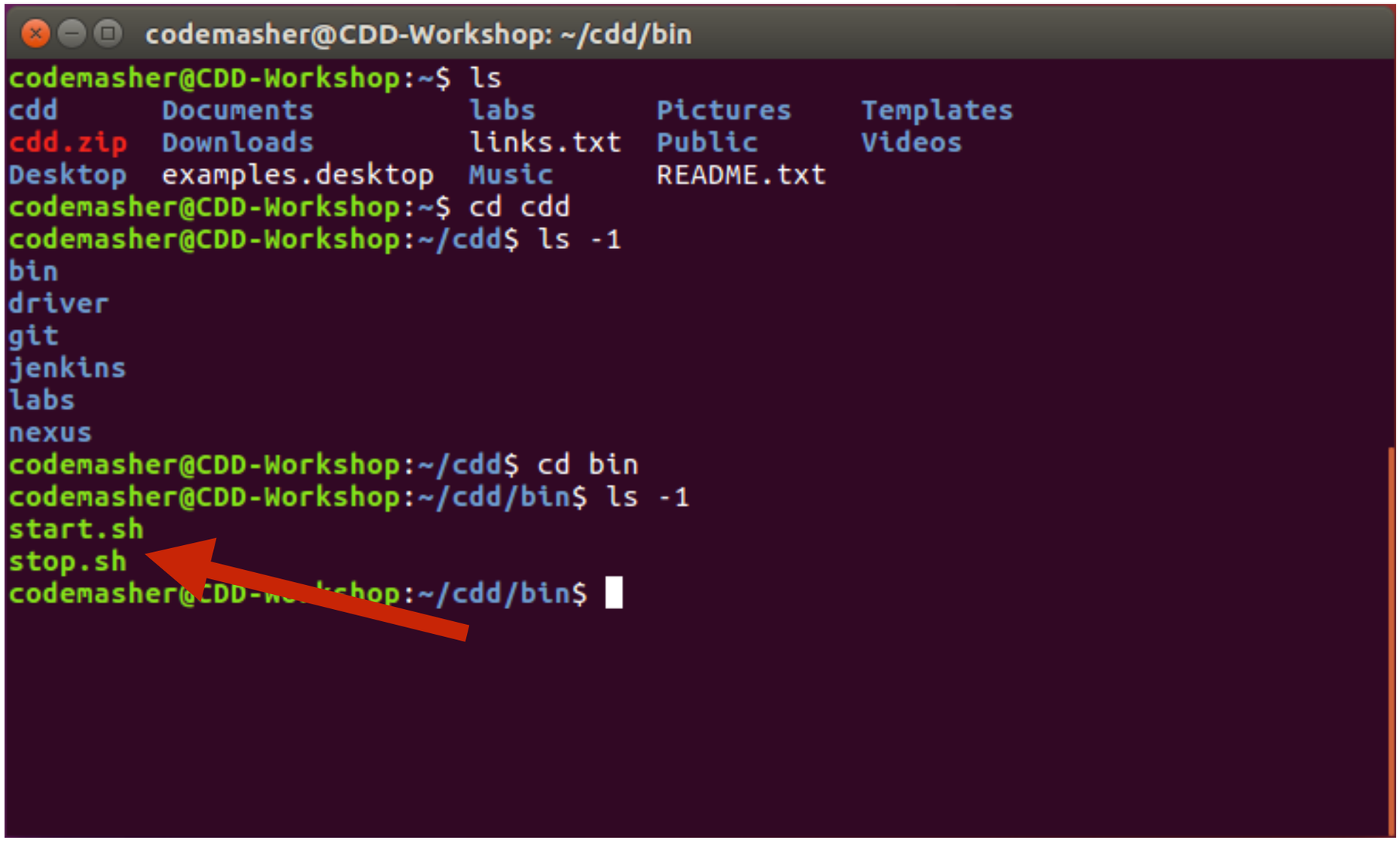



unzip cdd.zip

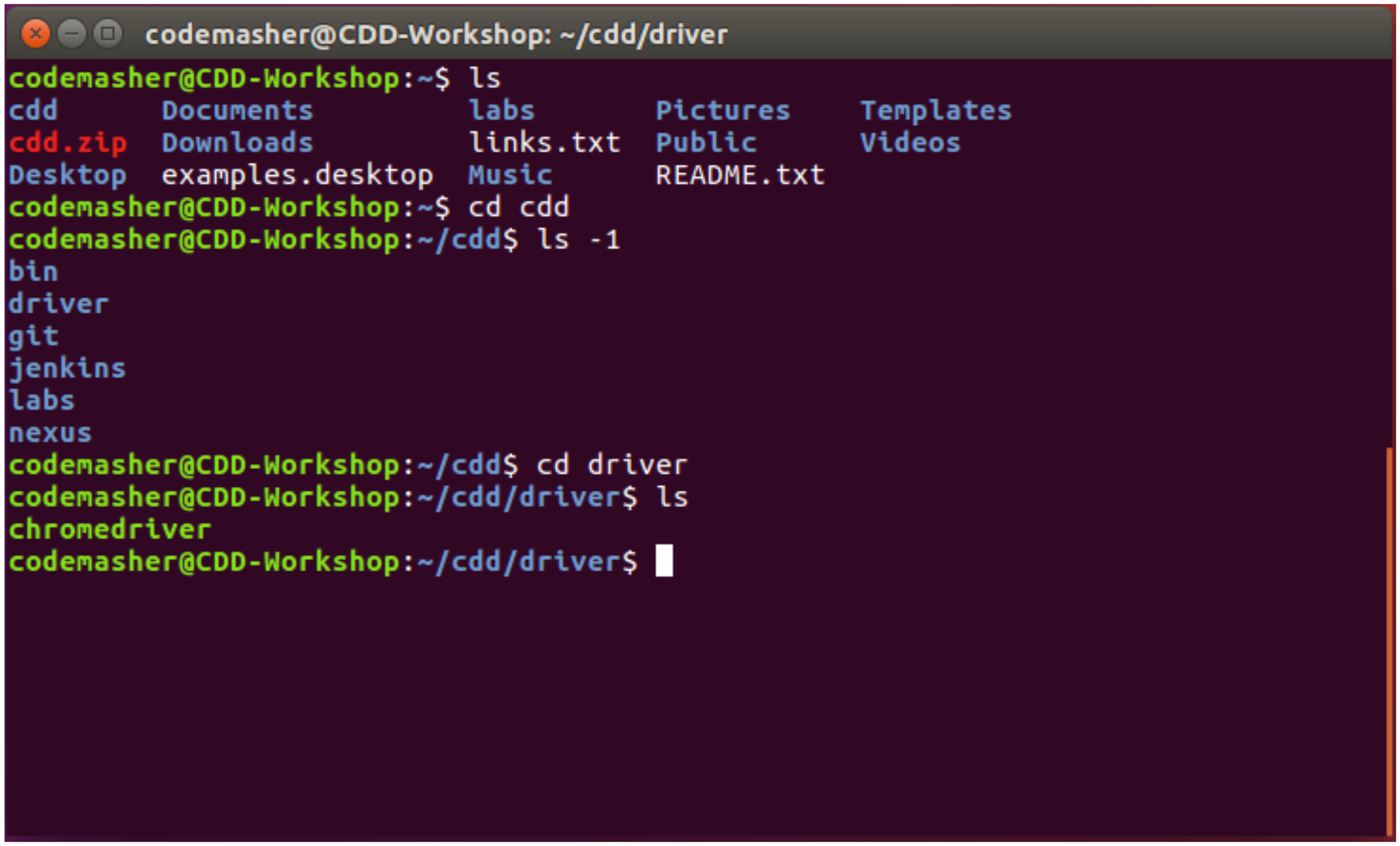

codemasher@CDD-Workshop: ~/cdd
codemasher@CDD-Workshop:~$ ls
cdd Documents labs Pictures Templates
cdd.zip Downloads links.txt Public Videos
Desktop examples.desktop Music README.txt
codemasher@CDD-Workshop:~$ cd cdd
codemasher@CDD-Workshop:~/cdd$ ls -1
bin
driver
git
jenkins
labs
nexus
codemasher@CDD-Workshop:~/cdd$




start.sh - starts Jenkins, Nexus, and git-daemon



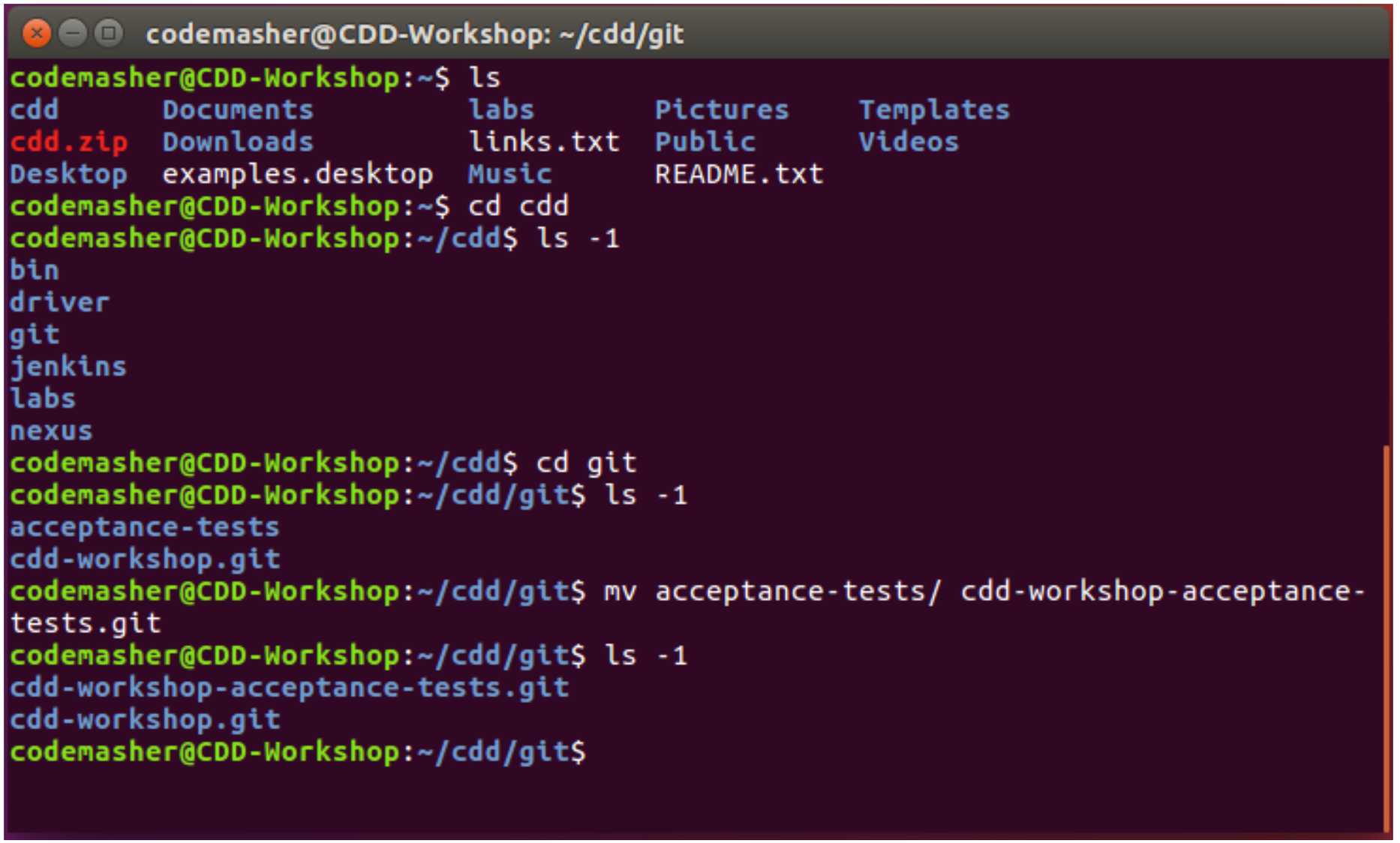

stop.sh - stops Jenkins, Nexus, and git-daemon

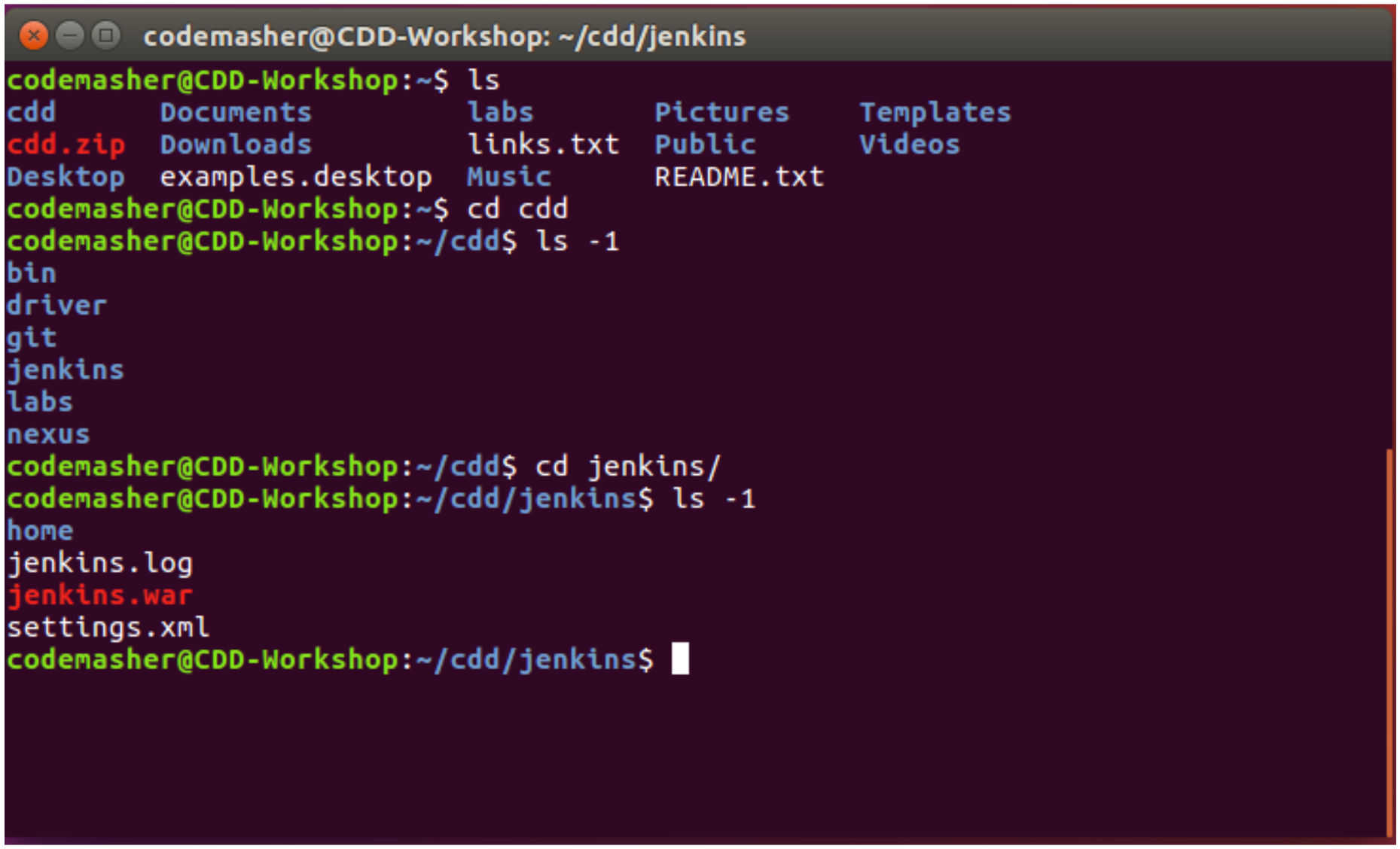

codemasher@CDD-Workshop: ~/cdd/driver
codemasher@CDD-Workshop:~$ ls
cdd Documents labs Pictures Templates
cdd.zip Downloads links.txt Public Videos
Desktop examples.desktop Music README.txt
codemasher@CDD-Workshop:~$ cd cdd
codemasher@CDD-Workshop:~/cdd$ ls -1
bin
driver
git
jenkins
labs
nexus
codemasher@CDD-Workshop:~/cdd$ cd driver
codemasher@CDD-Workshop:~/cdd/driver$ ls
chromedriver
codemasher@CDD-Workshop:~/cdd/driver$

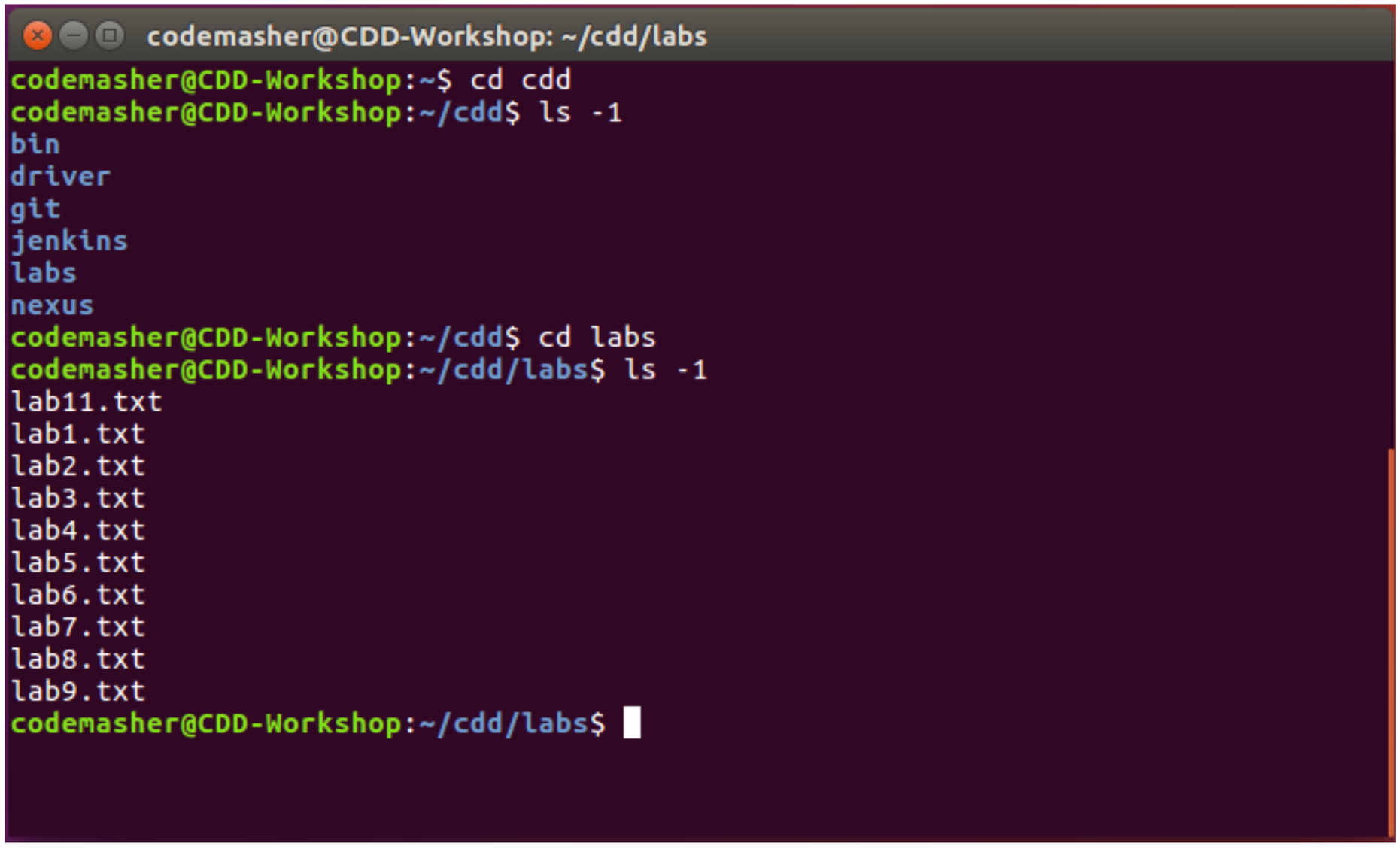
codemasher@CDD-Workshop: ~/cdd/git

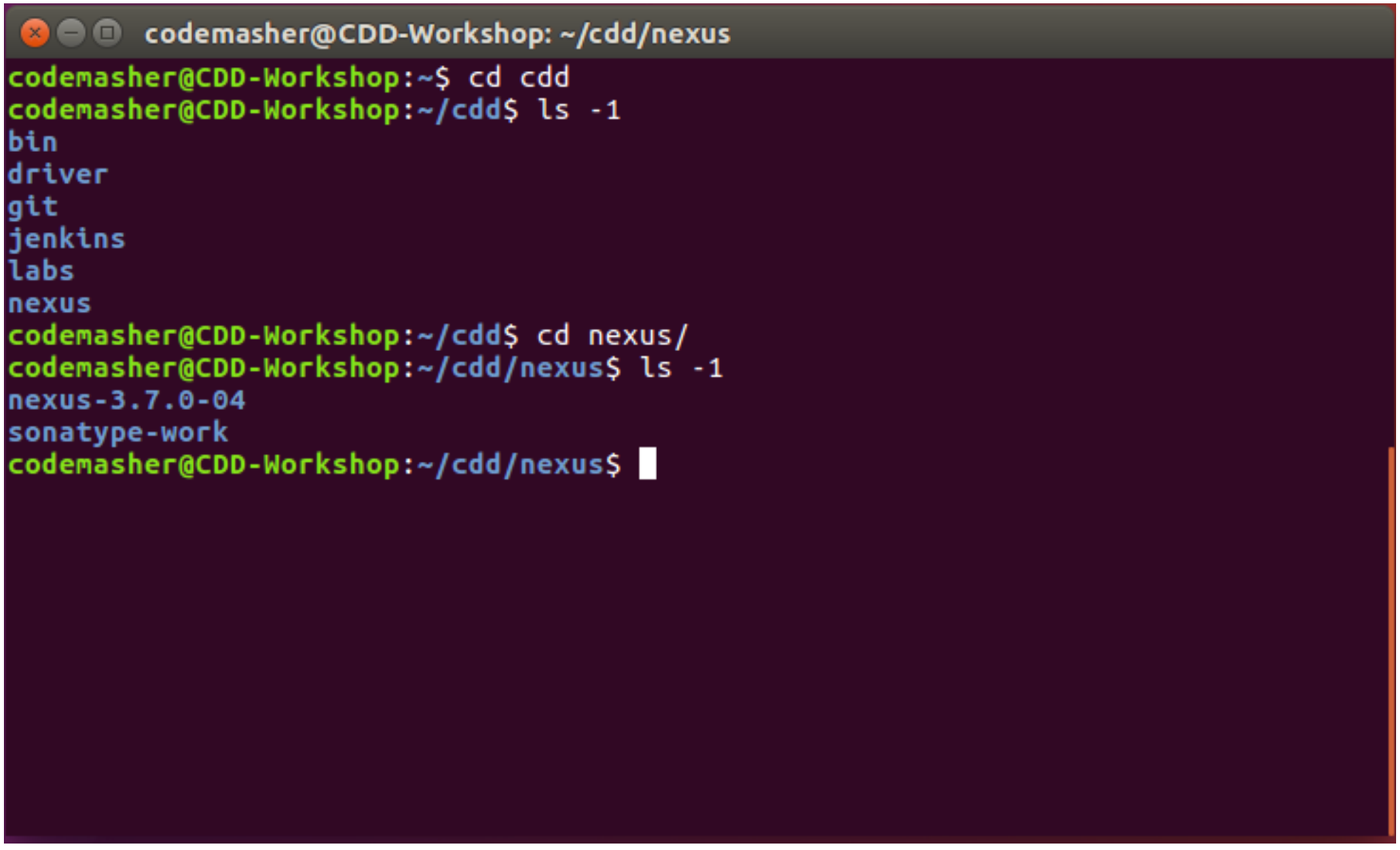
```
codemasher@CDD-Workshop:~$ ls
cdd       Documents          labs        Pictures    Templates
cdd.zip   Downloads          links.txt   Public      Videos
Desktop   examples.desktop   Music       README.txt
codemasher@CDD-Workshop:~$ cd cdd
codemasher@CDD-Workshop:~/cdd$ ls -1
bin
driver
git
jenkins
labs
nexus
codemasher@CDD-Workshop:~/cdd$ cd git
codemasher@CDD-Workshop:~/cdd/git$ ls -1
acceptance-tests
cdd-workshop.git
codemasher@CDD-Workshop:~/cdd/git$
```

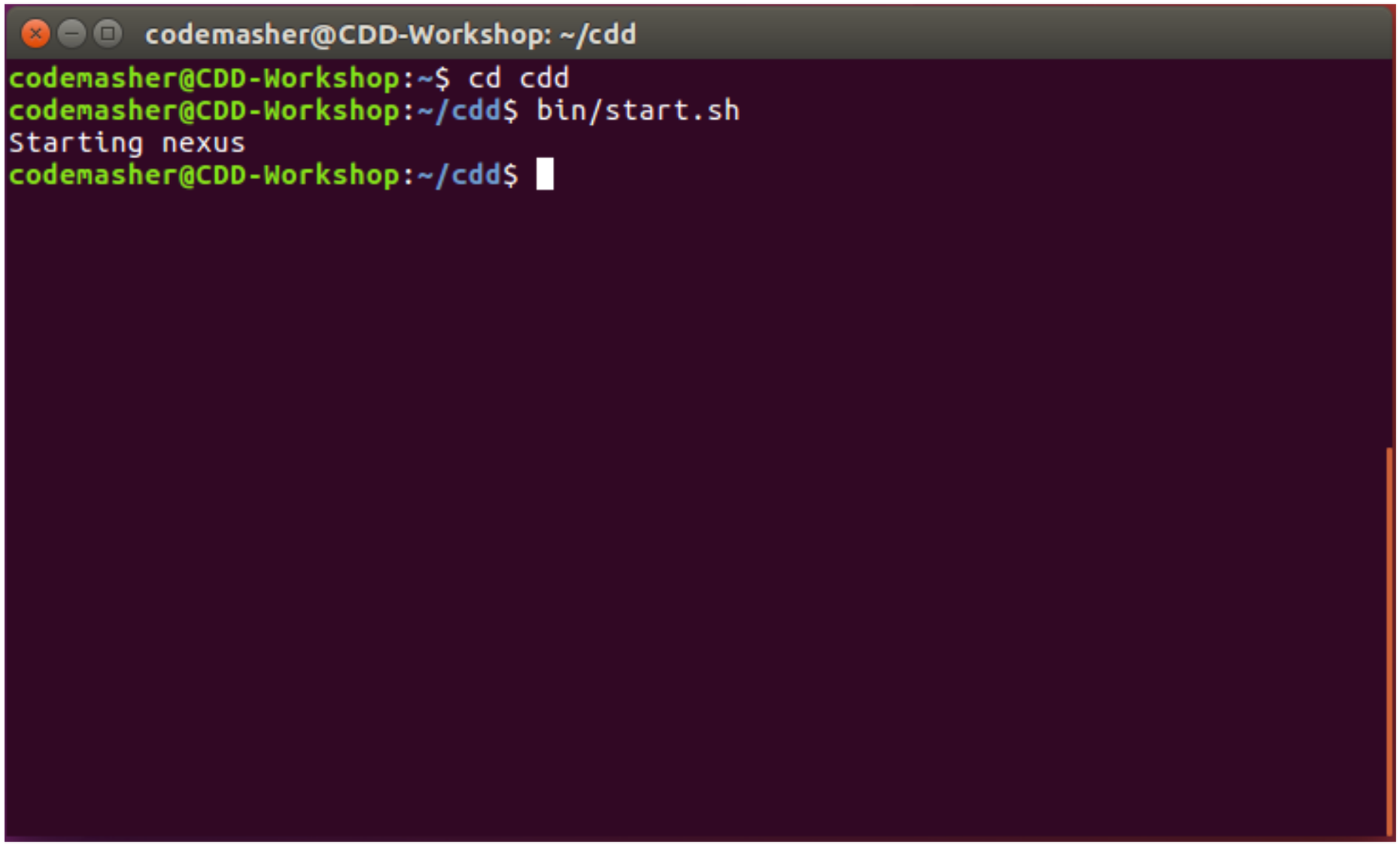
Need to fix typo:
mv acceptance-test cdd-workshop-acceptance-tests.git

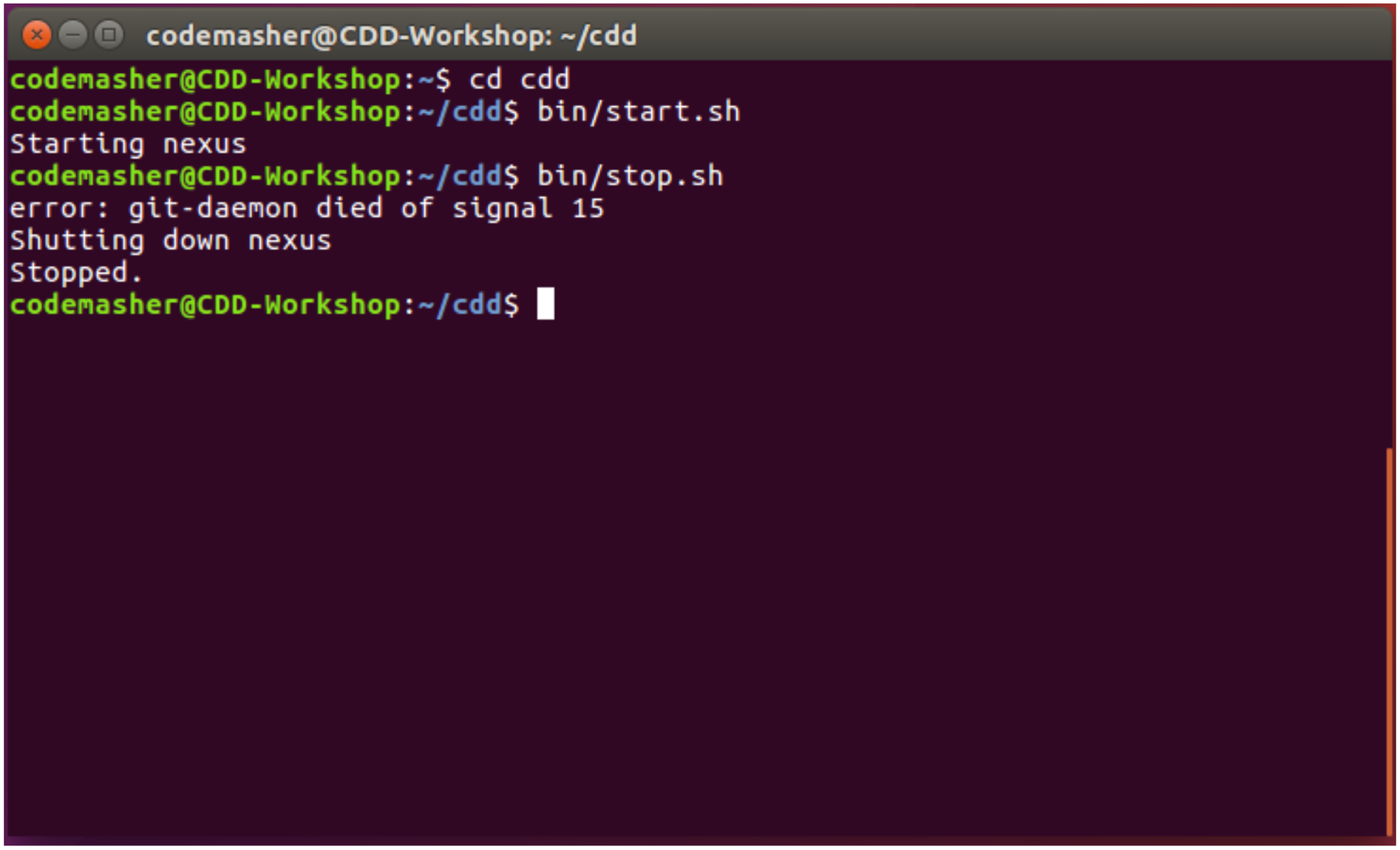

codemasher@CDD-Workshop: ~/cdd/git
codemasher@CDD-Workshop:~$ ls
cdd Documents labs Pictures Templates
cdd.zip Downloads links.txt Public Videos
Desktop examples.desktop Music README.txt
codemasher@CDD-Workshop:~$ cd cdd
codemasher@CDD-Workshop:~/cdd$ ls -1
bin
driver
git
jenkins
labs
nexus
codemasher@CDD-Workshop:~/cdd$ cd git
codemasher@CDD-Workshop:~/cdd/git$ ls -1
acceptance-tests
cdd-workshop.git
codemasher@CDD-Workshop:~/cdd/git$ mv acceptance-tests/ cdd-workshop-acceptance-tests.git
codemasher@CDD-Workshop:~/cdd/git$ ls -1
cdd-workshop-acceptance-tests.git
cdd-workshop.git
codemasher@CDD-Workshop:~/cdd/git$

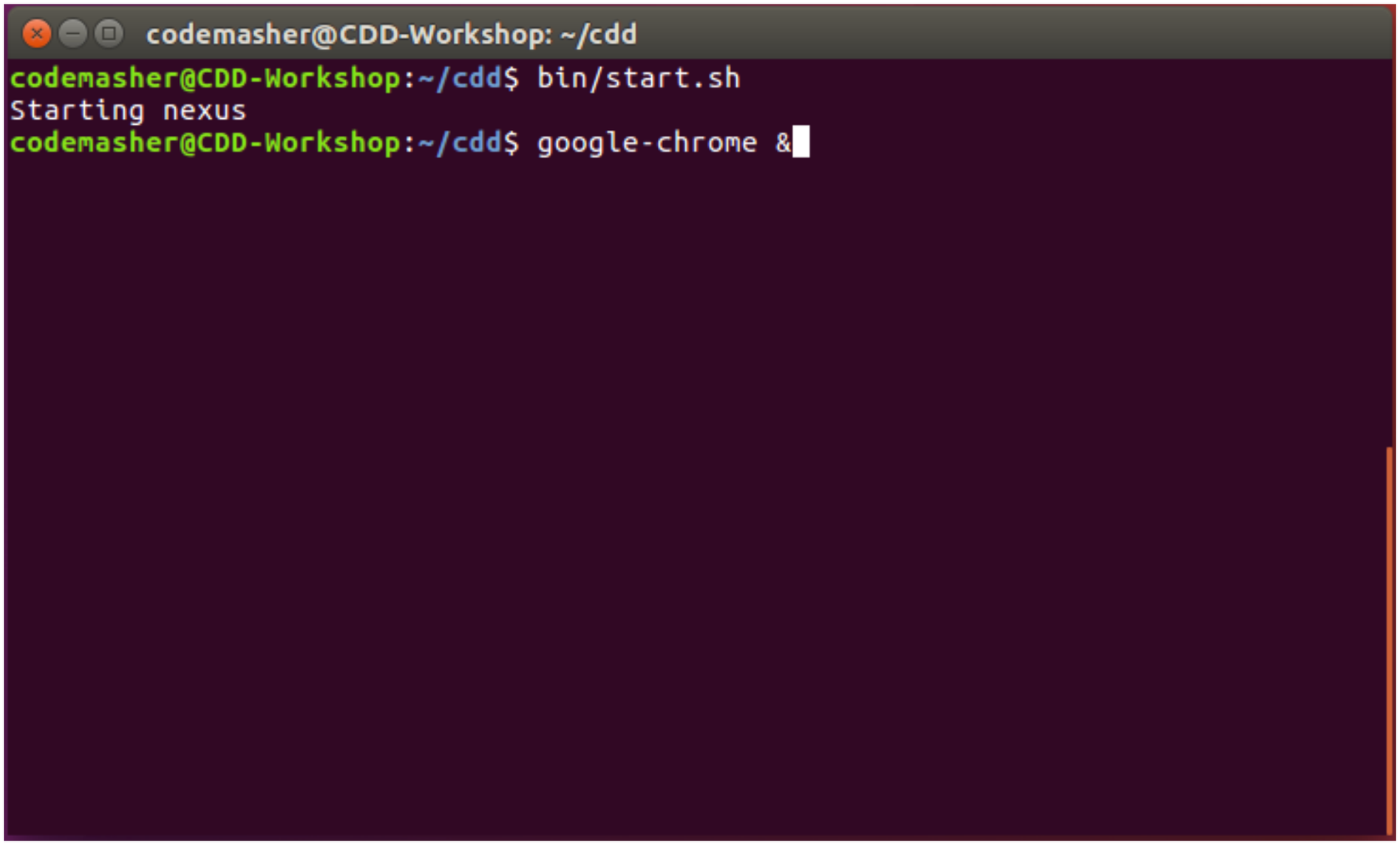

codemasher@CDD-Workshop: ~/cdd/jenkins
codemasher@CDD-Workshop:~$ ls
cdd  Documents  labs  Pictures  Templates
cdd.zip  Downloads  links.txt  Public  Videos
Desktop  examples.desktop  Music  README.txt
codemasher@CDD-Workshop:~$ cd cdd
codemasher@CDD-Workshop:~/cdd$ ls -1
bin
driver
git
jenkins
labs
nexus
codemasher@CDD-Workshop:~/cdd$ cd jenkins/
codemasher@CDD-Workshop:~/cdd/jenkins$ ls -1
home
jenkins.log
jenkins.war
settings.xml
codemasher@CDD-Workshop:~/cdd/jenkins$

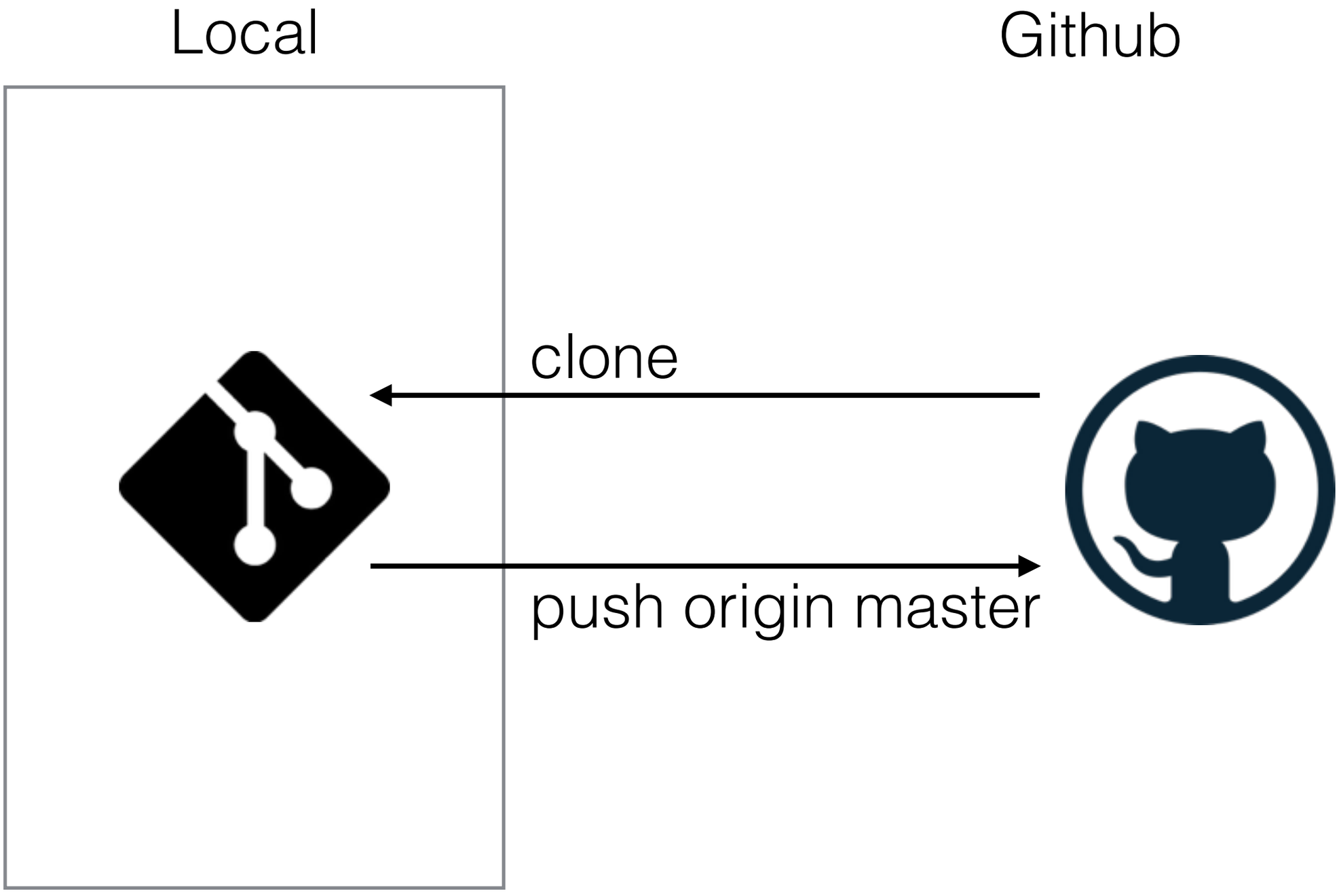

codemasher@CDD-Workshop: ~/cdd/labs
codemasher@CDD-Workshop:~$ cd cdd
codemasher@CDD-Workshop:~/cdd$ ls -1
bin
driver
git
jenkins
labs
nexus
codemasher@CDD-Workshop:~/cdd$ cd labs
codemasher@CDD-Workshop:~/cdd/labs$ ls -1
lab11.txt
lab1.txt
lab2.txt
lab3.txt
lab4.txt
lab5.txt
lab6.txt
lab7.txt
lab8.txt
lab9.txt
codemasher@CDD-Workshop:~/cdd/labs$

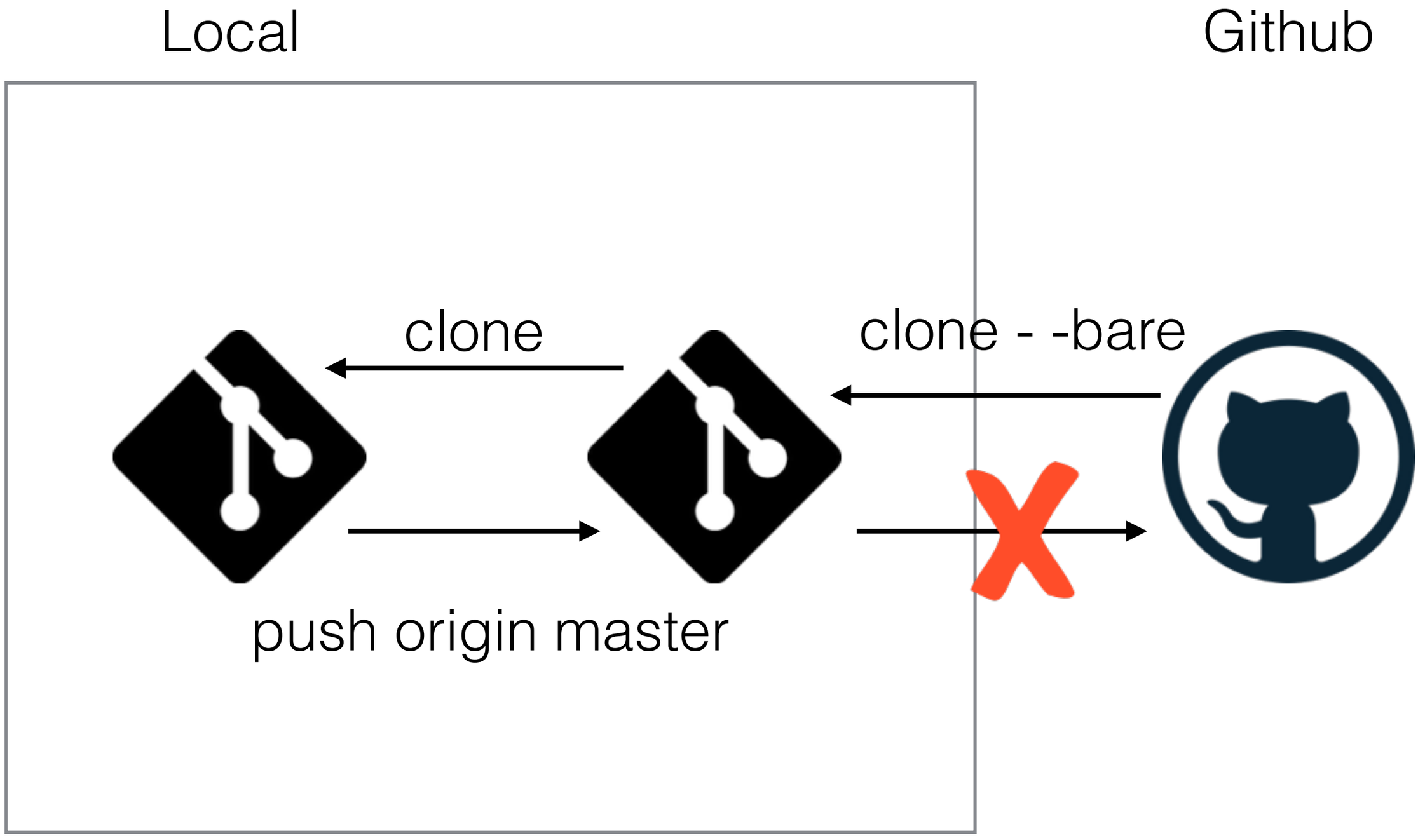

codemasher@CDD-Workshop: ~/cdd/nexus
codemasher@CDD-Workshop:~$ cd cdd
codemasher@CDD-Workshop:~/cdd$ ls -1
bin
driver
git
jenkins
labs
nexus
codemasher@CDD-Workshop:~/cdd$ cd nexus/
codemasher@CDD-Workshop:~/cdd/nexus$ ls -1
nexus-3.7.0-04
sonatype-work
codemasher@CDD-Workshop:~/cdd/nexus$


codemasher@CDD-Workshop: ~/cdd
codemasher@CDD-Workshop:~$ cd cdd
codemasher@CDD-Workshop:~/cdd$ bin/start.sh
Starting nexus
codemasher@CDD-Workshop:~/cdd$


codemasher@CDD-Workshop: ~/cdd
codemasher@CDD-Workshop:~$ cd cdd
codemasher@CDD-Workshop:~/cdd$ bin/start.sh
Starting nexus
codemasher@CDD-Workshop:~/cdd$ bin/stop.sh
error: git-daemon died of signal 15
Shutting down nexus
Stopped.
codemasher@CDD-Workshop:~/cdd$


codemasher@CDD-Workshop: ~/cdd
codemasher@CDD-Workshop:~/cdd$ bin/start.sh
Starting nexus
codemasher@CDD-Workshop:~/cdd$ google-chrome &

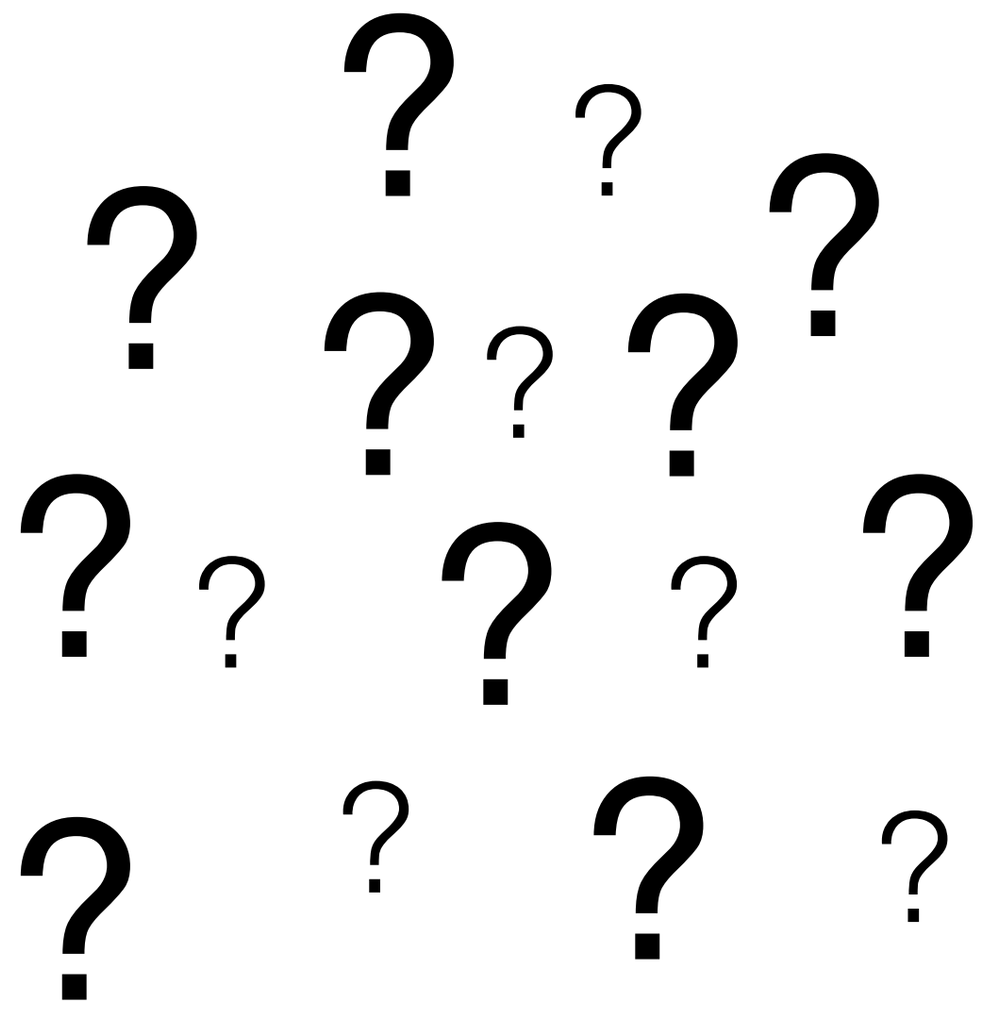

Local
Github
clone
push origin master


Local
Github
clone
clone - -bare
push origin master

# Meet The Team

# Sam

Tech lead



Diane
Developer

Quinn
QA Lead

Todd
Tech ops

# Sam automates the build

# What is our build process?

What is our
build process?

We use

What is our
build process?

We start the build
after we commit code.

What is our
build process?

The build creates a
standalone jar.

What is our
build process?

We start a deploy
job to push to QA.

What is our
build process?

We test and approve
or reject the build.

Hmmm…

Could we start the build after each commit?

# Lab 1:

# Create pipeline with build stage

```
pipeline {

}
```

```
pipeline {
    agent any

}
```

```
pipeline {
    agent any

    tools {


    }

}
```

```
pipeline {
    agent any

    tools {
        jdk 'JDK'

    }
}
```

```
pipeline {
   agent any

   tools {
      jdk 'JDK'
      maven 'M3'
   }

}
```

```
pipeline {
   agent any

   tools {
      jdk 'JDK'
      maven 'M3'
   }

   stages {


   }
}
```

```
pipeline {
   agent any

   tools {
      jdk 'JDK'
      maven 'M3'
   }

   stages {
      stage(…){
      }
   }
}
```

```
stage('Build') {

}
```

```
stage('Build') {
    steps {

    }
}
```

```
stage('Build') {
   steps {
      configFileProvider(

                                )
      }
}
```

```
stage('Build') {
   steps {
      configFileProvider(
         [

                                    ])
      }
}
```

```
stage('Build') {
   steps {
      configFileProvider(
         [configFile(

                                 )])
      }
}
```

```
stage('Build') {
   steps {
      configFileProvider(
         [configFile(fileId:
         ‘maven-settings’,
                              )])
      }
}
```

```
stage('Build') {
   steps {
      configFileProvider(
         [configFile(fileId:
         ‘maven-settings’, variable:
         'MAVEN_SETTINGS')])
      }
}
```

```
stage('Build') {
   steps {
      configFileProvider(
         [configFile(fileId:
         ‘maven-settings’, variable:
         'MAVEN_SETTINGS')]) {
         }
      }
}
```

```
stage('Build') {
   steps {
      configFileProvider(
         [configFile(fileId:
         ‘maven-settings’, variable:
         'MAVEN_SETTINGS')]) {
         sh '
                                  '
         }
      }
}
```

```
stage('Build') {
   steps {
      configFileProvider(
         [configFile(fileId:
         ‘maven-settings’, variable:
         'MAVEN_SETTINGS')]) {
        sh 'mvn
                                  '
         }
      }
}
```

```
stage('Build') {
   steps {
      configFileProvider(
         [configFile(fileId:
         ‘maven-settings’, variable:
         'MAVEN_SETTINGS')]) {
        sh 'mvn -B
                                    '
         }
      }
}
```

```
stage('Build') {
  steps {
    configFileProvider(
        [configFile(fileId:
        ‘maven-settings’, variable:
        'MAVEN_SETTINGS')]) {
      sh 'mvn -B -s $MAVEN_SETTINGS
                                    '
        }
    }
}
```

```
stage('Build') {
   steps {
      configFileProvider(
         [configFile(fileId:
         ‘maven-settings’, variable:
         'MAVEN_SETTINGS')]) {
        sh 'mvn -B -s $MAVEN_SETTINGS
            clean
         }
      }
}
```

```
stage('Build') {
  steps {
    configFileProvider(
        [configFile(fileId:
        ‘maven-settings’, variable:
        'MAVEN_SETTINGS')]) {
      sh 'mvn -B -s $MAVEN_SETTINGS
          clean package                '
    }
  }
}
```

```
stage('Build') {
  steps {
    configFileProvider(
        [configFile(fileId:
        ‘maven-settings’, variable:
        'MAVEN_SETTINGS')]) {
      sh 'mvn -B -s $MAVEN_SETTINGS
        clean package -DskipTests'
    }
  }
}
```

```
stage('Build') {
  steps {
    configFileProvider(
        [configFile(fileId:
        ‘maven-settings’, variable:
        'MAVEN_SETTINGS')]) {
      sh 'mvn -B -s $MAVEN_SETTINGS
        clean package -DskipTests'
    }

    archiveArtifacts

  }
}
```

```
stage('Build') {
  steps {
    configFileProvider(
      [configFile(fileId:
      ‘maven-settings’, variable:
      'MAVEN_SETTINGS')]) {
      sh 'mvn -B -s $MAVEN_SETTINGS
        clean package -DskipTests'
    }

    archiveArtifacts artifacts:
      '**/target/*.jar'
  }
}
```

# Sam adds versioning to the the build

Hello?

Hello?

The application is not responding!

I think I know what’s happening.

The wrong version of the app was deployed.

I’ll deploy the correct one now.

There!  It’s working now.

Confirmed.  The app is behaving normally now.

Hmmm…

How can we know we are deploying the right jar?

How can we know we are deploying the right jar?

We could version each build that we give to QA.

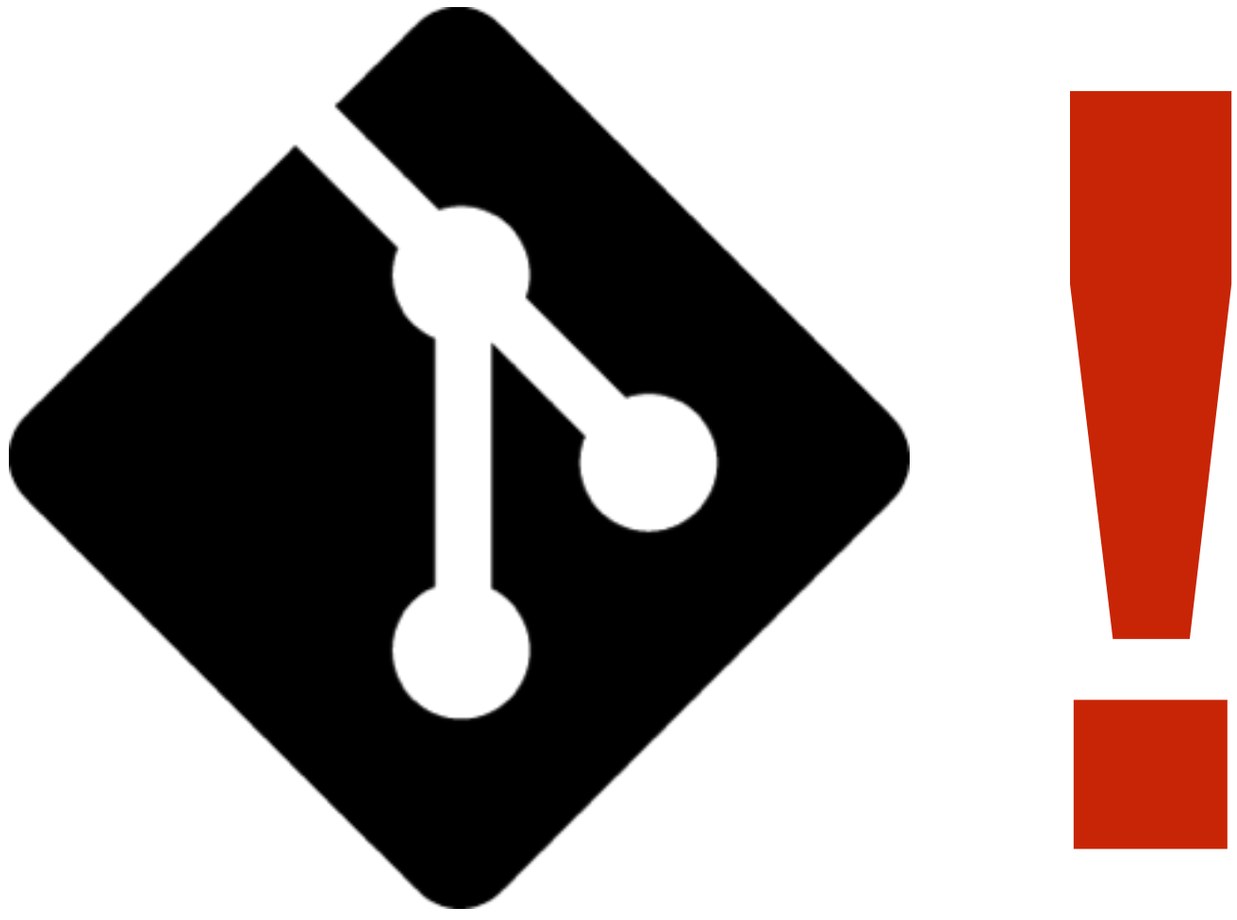
And we could tag revisions in source control.

We could version each build that we give to QA.

# Lab 2:

# Increment version and tag

- Update ‘Build’ stage
  - Use “ci/updateGitForPush.sh” to prepare git to accept changes
  - Use “ci/incrementPomVersion.sh” to update version
- Create new stage: ‘Tag’, placed after ‘Build’ stage
  - Use “ci/commitPomVersion.sh” to make version update permanent in pom.xml
  - Use “ci/tag.sh to update tag in git



Sam pushes artifacts to Nexus

Hello

Hello

It’s good that we’re tagging builds, but…

Hello

We need to be able to prove which artifact we are deploying.

# Why is that?

We need to be able to prove which artifact we are deploying.

Why is that?

Auditing.  Artifacts must be stored where developers only have read access.

We could push to a Nexus repo as part of the build.

Auditing.  Artifacts must be stored where developers only have read access.

# Lab 3:

# Push artifacts to Nexus

- Add new stage after ‘Tag’
- Give it a descriptive name
- ‘mvn deploy’ will push to Nexus
- Skip tests
- Remember to supply ‘settings.xml’ file

# Sam adds tests to the build

Hello?

Tech Ops

Hello?

The application is not responding again!

Tech Ops

I think I’ve figured it out. I’ll need to deploy a fix.

All done.  The app is back up.

Confirmed.  Everything is back to normal.

Hmmm…

# How do we reduce the risk of bugs making it into production?

How do we reduce the risk of bugs making it into production?

Well, we have unit tests…

How do we reduce the risk of bugs making it into production?

but some of them are broken.

How do we reduce the risk of bugs making it into production?

We run them locally…

How do we reduce the risk of bugs making it into production?

but, we ignore the failures.

Let’s fix or remove any broken tests.

Let’s fix or remove any broken tests.

And we can run the tests as a part of each build.

Can we pair up with QA to automate tests of our most critical features?

Can we pair up with QA to automate tests of our most critical features?

Sure. We can do one scenario per sprint.

# Lab 4:

# Add tests to the pipeline

- Add new stage for testing
- 'mvn test’ will run unit tests
- 'mvn failsafe:integration-test’ will run integration tests
- Remember to supply ‘settings.xml’ file

# Sam adds health checks to the app

# Who detected the outages?

Who detected the outages?

Customers called in because they couldn’t access the site.

Hmmm…

Can we detect issues before our customers?

Can we detect issues before our customers?

We could add health checks and monitors.

And we could use the health checks to detect bad deployments before they go live.

We could add health checks and monitors.

# Lab 5:

# Deploy test instance with health check

- Add new stage for to test env
- Use deploy.sh to deploy with health check
- Deploy to port 8091

# Sam adds acceptance tests to the build

I analyzed the last issue that caused an outage.

I analyzed the last issue that caused an outage.

What did you find out?

Our unit and integration tests wouldn’t have caught it.

What did you find out?

Our unit and integration tests wouldn’t have caught it.

It’s time to start running our acceptance test suite with each build.

# Thanks QA!!!!

Thanks QA!!!!

No problem. Let’s see those tests in action.

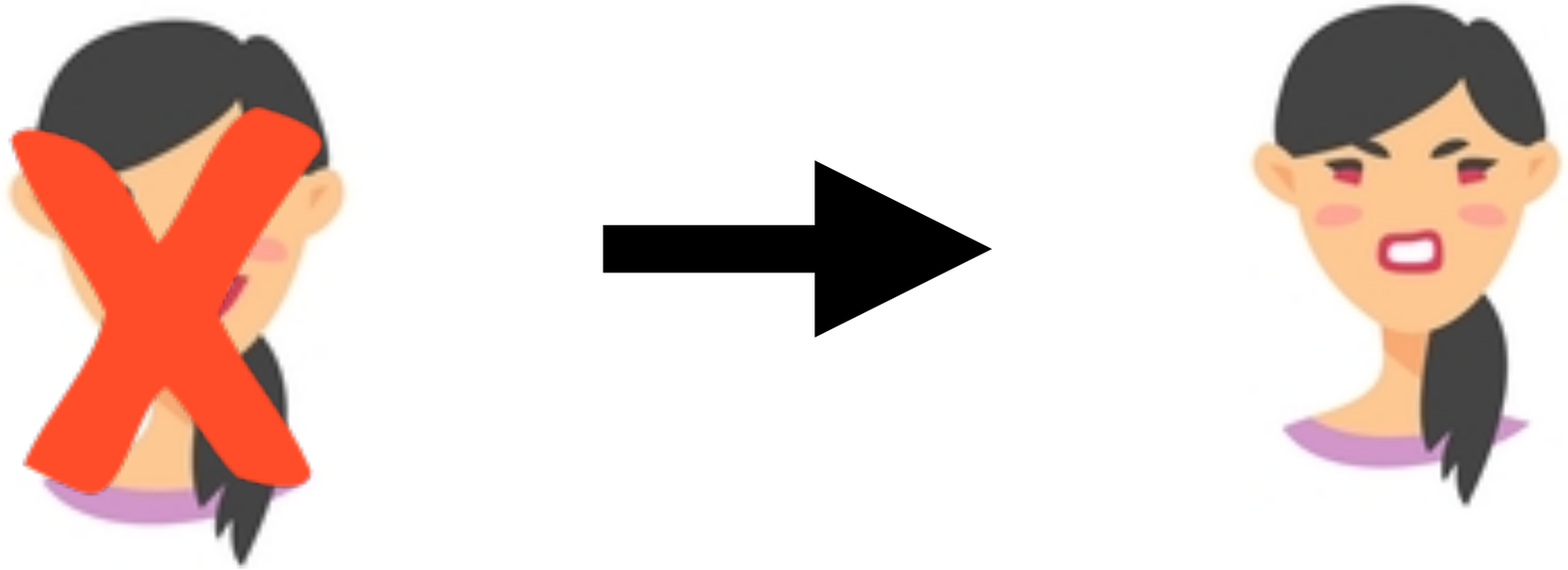
# Lab 6:

# Add acceptance tests

- Add new stage for to acceptance tests after deploy stage
- Checkout acceptance test project using ‘ci/checkoutAcceptanceTests.sh’
- After checking out code, ‘cd acceptance-tests’
- From the acceptance test dir, run ‘mvn clean install -Dport=8091'

# An aside about QA…


JAR




JAR


JAR

# Sam adds parallel tests to the build

The builds are slowing down.

We keep adding integration and acceptance tests.

I’ve looked at the tests and the ones we have are all appropriate.

What can we do make them run faster?

Hmmm…

What if we partition the tests and run them in parallel?

# Lab 7:

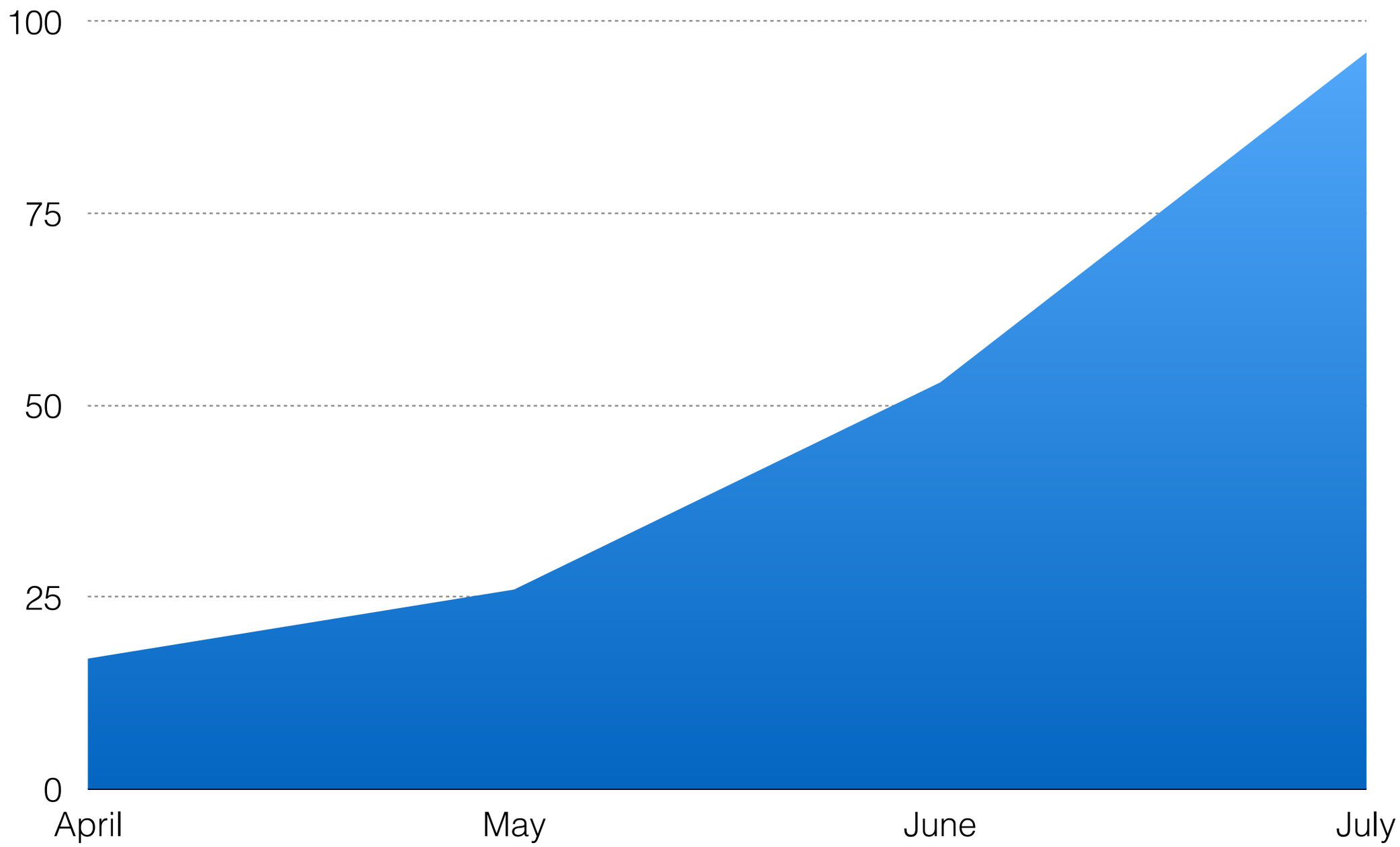
# Add parallel tests

- Combine unit, integration, and acceptance tests into one phase
- Use parallel function to run tests (see example)
- Make sure test deploy and acceptance stage are combined into one parallel phase

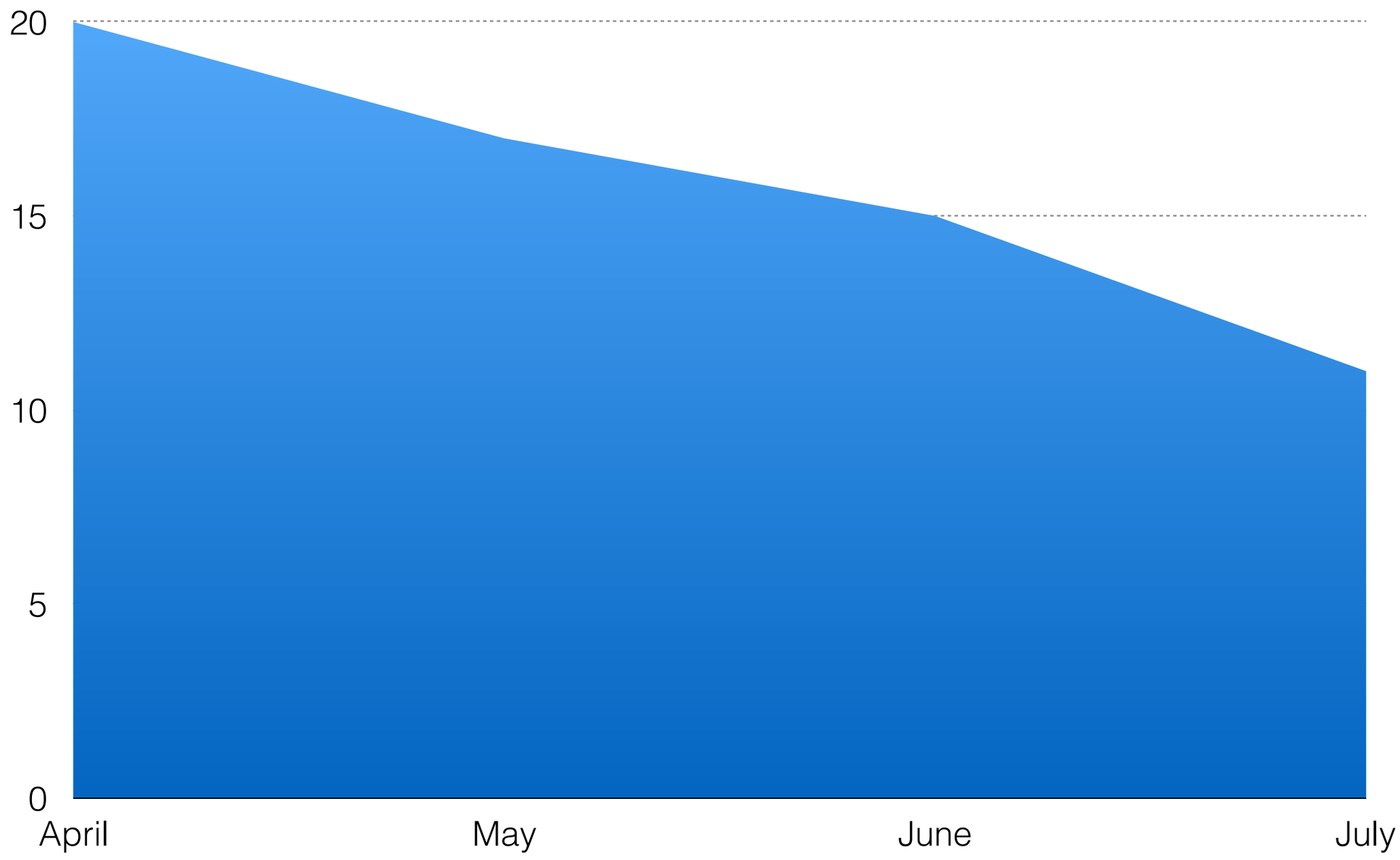
```
stage (‘Tests’) {
   steps {
      parallel( name1:{
         <commands>
      },
      name2: {
        <commands>
      })
   }
}
```

# Sam automates deployments to production

# Code Coverage



# Code Complexity



It has been 163 days since our last deployment failure.

The team has really stepped up.  We're ready.

The team has really stepped up.  We're ready.

Ready for what?

Continuous deployment.

Ready for what?

# Caution!

## There be dragons here!

The business should control when we deploy.

The business should control when we deploy.

You’re right, but you can still have control.

How?

You’re right, but you can still have control.

# How?

We will deploy our changes as soon as they pass quality checks, but…

How?

You decide when changes are activated in production.

So, I control when features go live like I've always done?

You decide when changes are activated in production.

So, I control when features go live like I've always done?

That's right!

# Lab 8:

# Deploy to production

- Create new stage at end of pipeline
- Deploy to “prod/blue”, port 10001
- Deploy to “prod/green”, port 10002

# Sam adds feature toggles to the app

I’ve noticed our stories are taking a long time to develop.

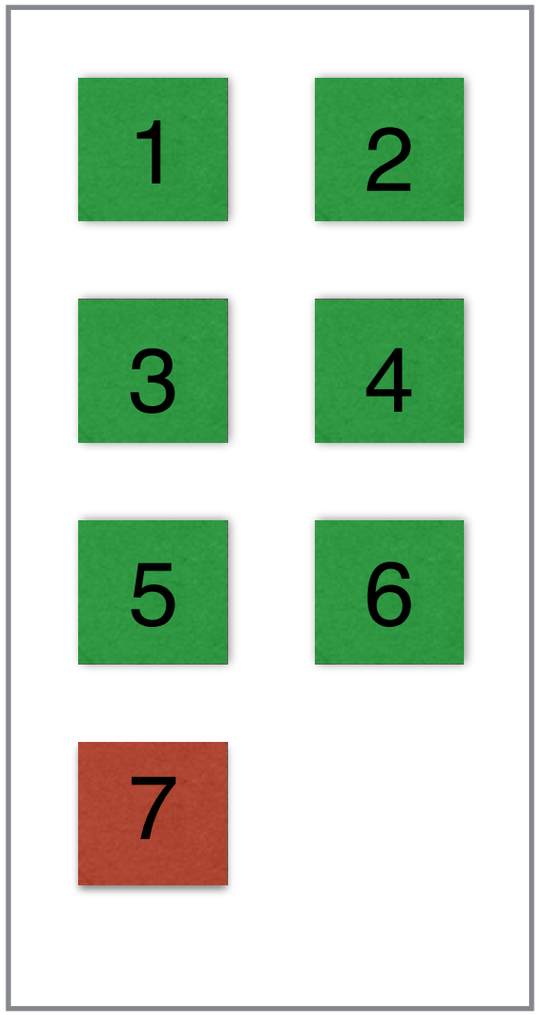
We break them down to the smallest feature that delivers value.

Why can’t we make stories smaller than a feature?

Well, how would we deploy partial features?

Hmmm…

# Feature Flags




1 = active

2 = active

3 = active

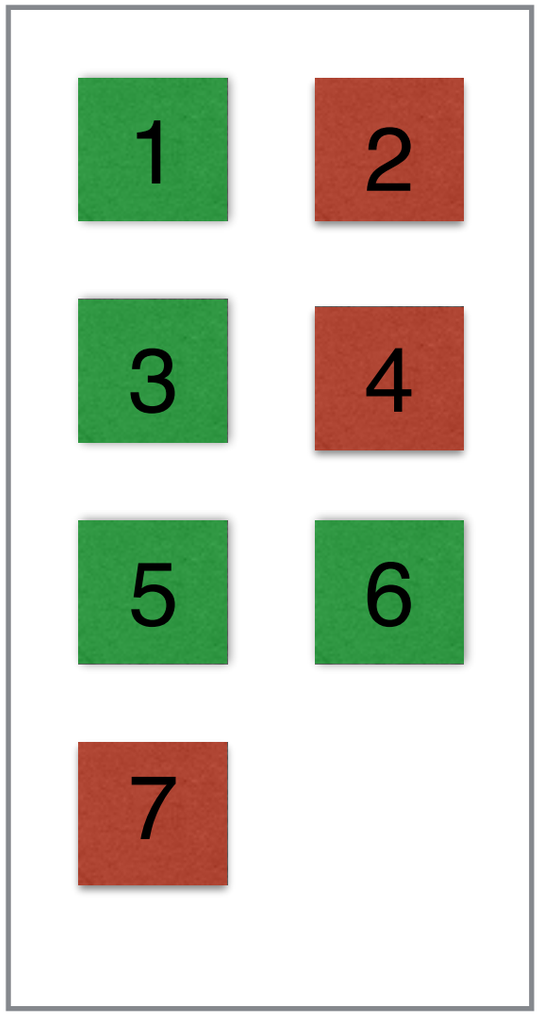
4 = active

5 = active

6 = active

7 = not active

# Feature Flags




1 = active

2 = not active

3 = active

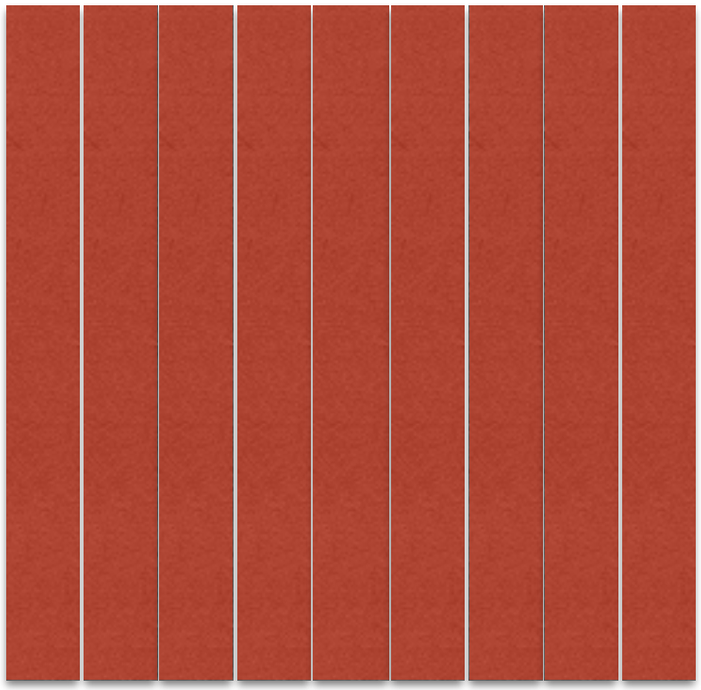
4 = not active

5 = active

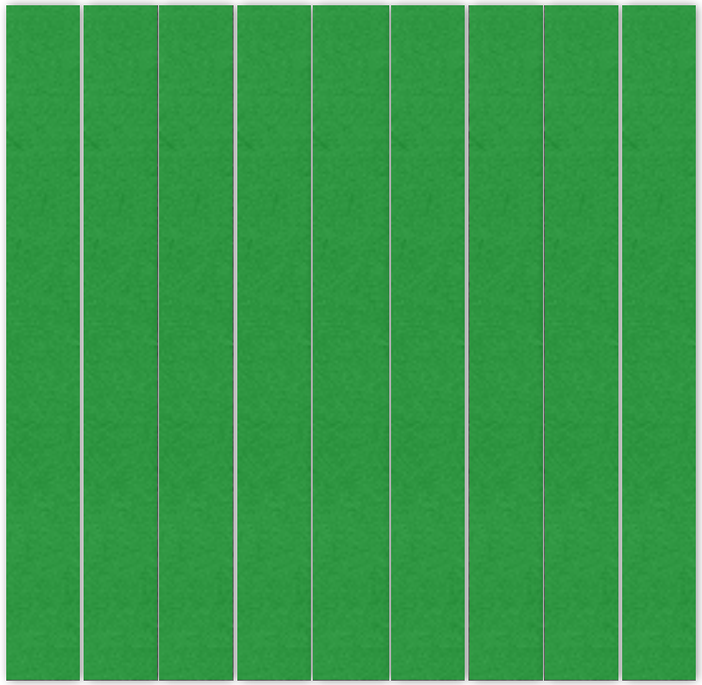
6 = active

7 = not active

# Feature Complete!



# Feature Activated!



# Sam uses feature toggles for a/b testing

# How can I help?

How can I help?

We can’t agree on how to build this feature.

How can I help?

Which way should we do it?

Hmmm…

Which way should we do it?

We could implement the feature both ways.

Which way should we do it?

We could implement the feature both ways.

How?

We’ll leverage feature toggles to switch between implementations.

How?

We'll leverage feature toggles to switch between implementations.

How will we know which feature to keep?

We’ll gather metrics of each feature from users and evaluate.

How will we know which feature to keep?

# Feature Toggle

# Demo

# Sam adds blue-green deployments

$10,000

$100,000

Business Partner

Oh?  I'll talk to my team.

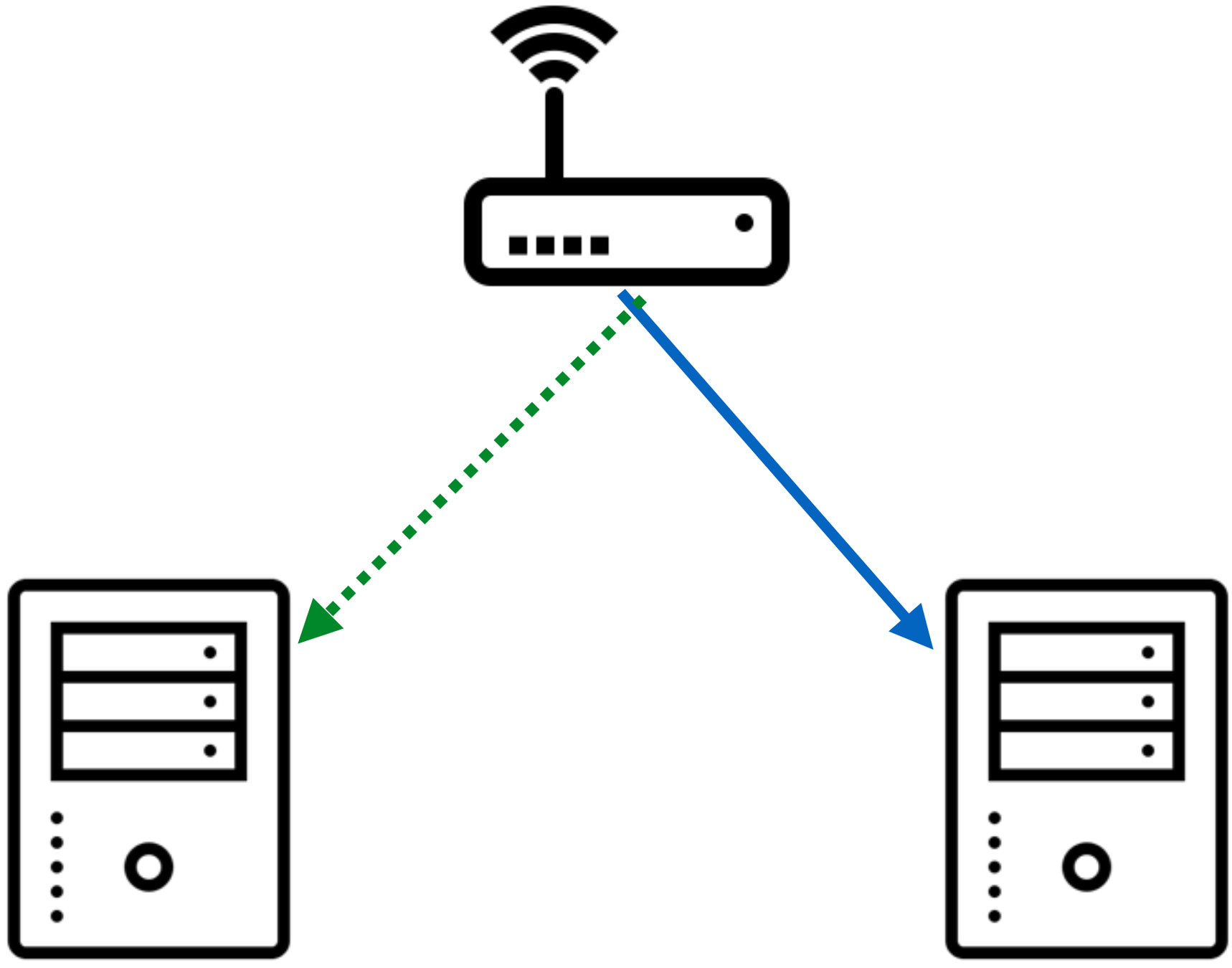
Hmmm…

We need to prevent downtime during our deployments.

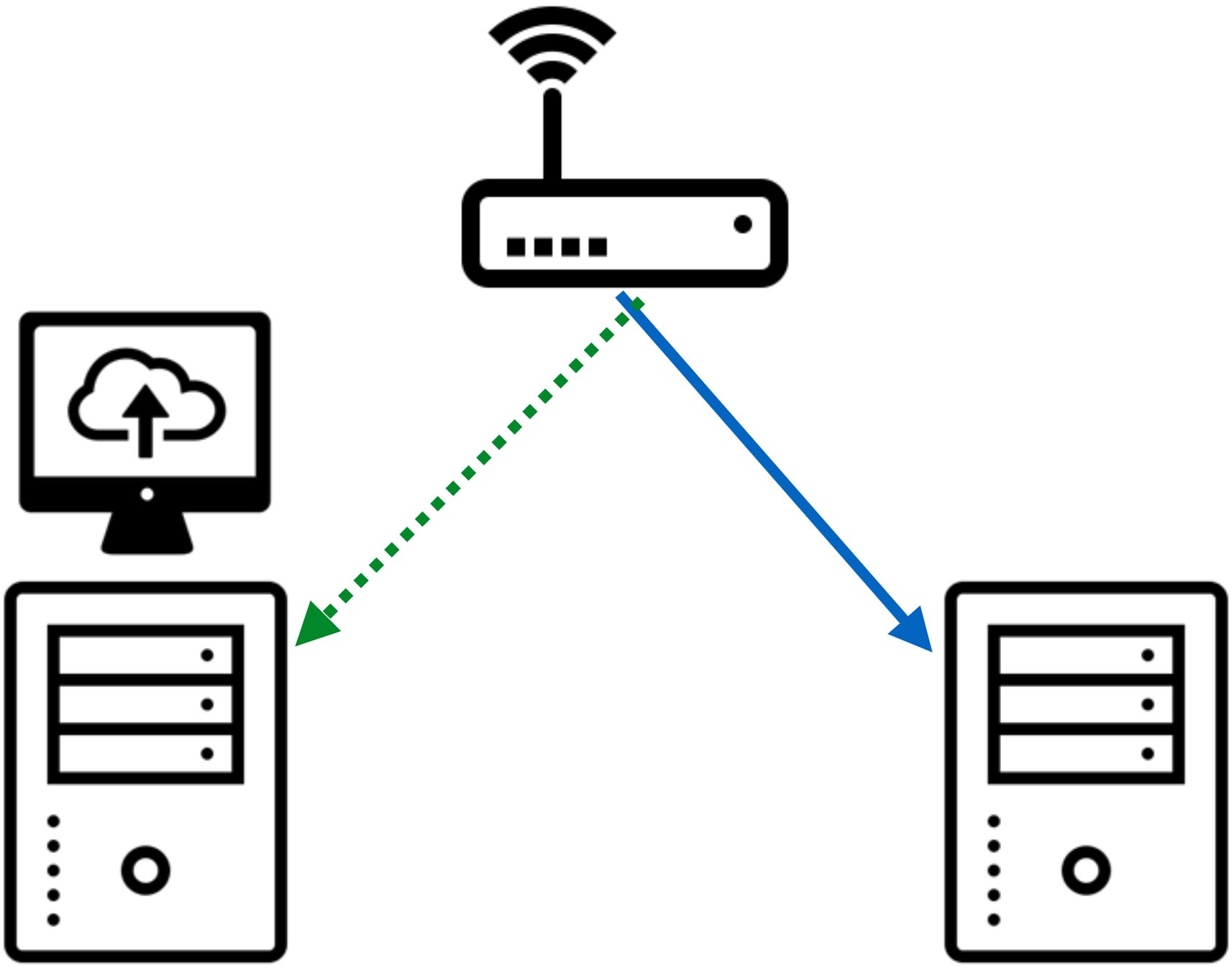
We need to prevent downtime during our deployments.

How?

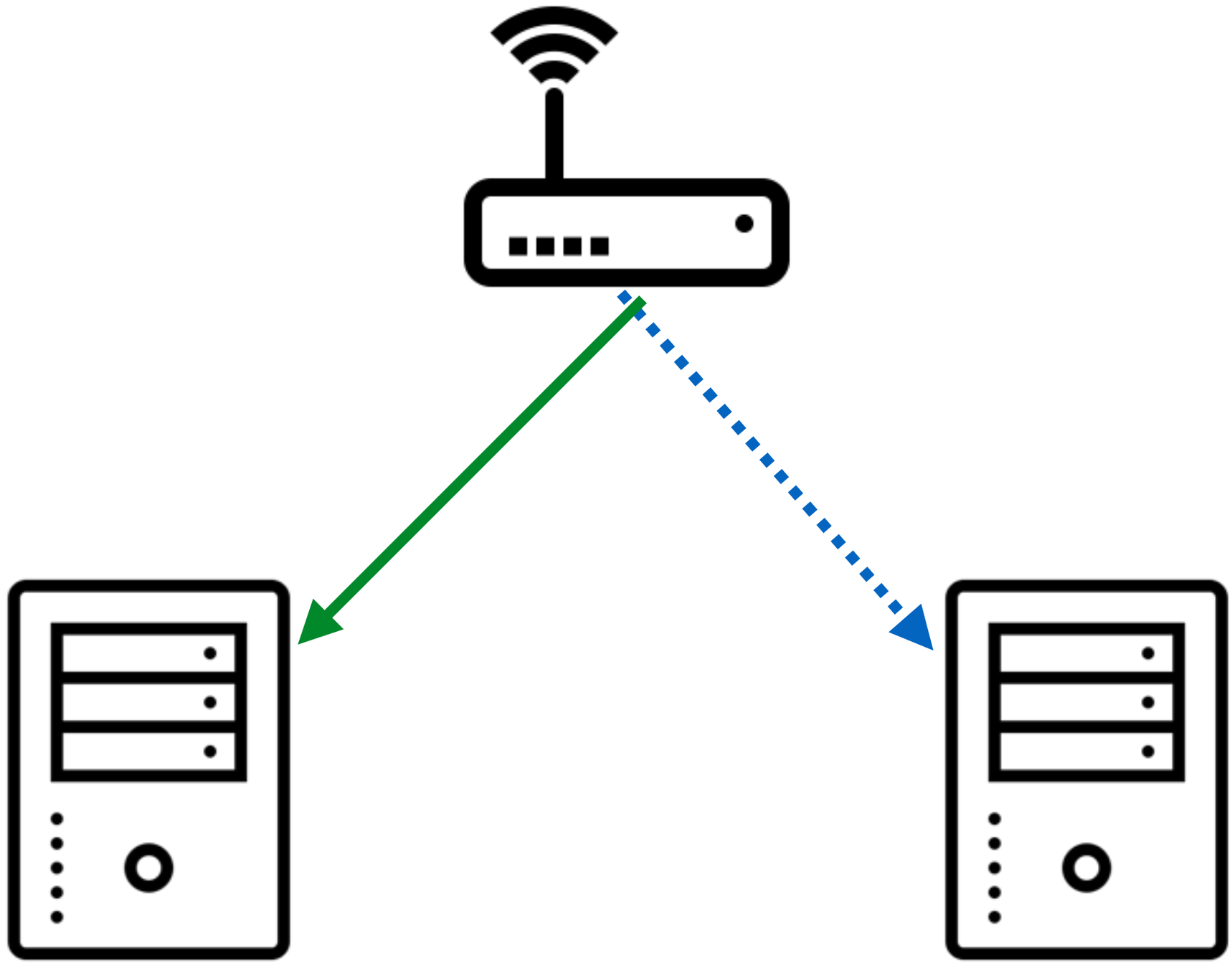
# Blue-green



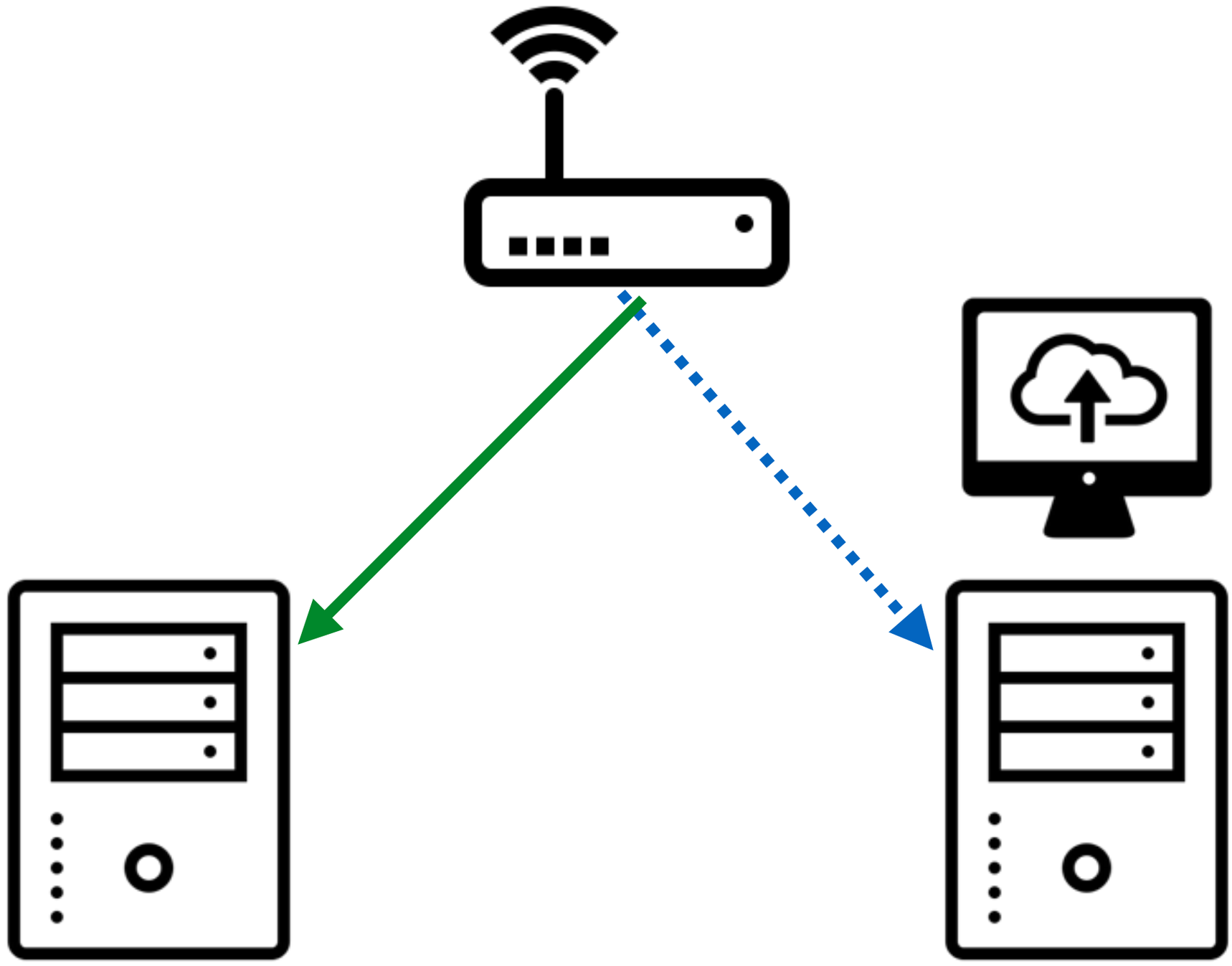
# Blue-green



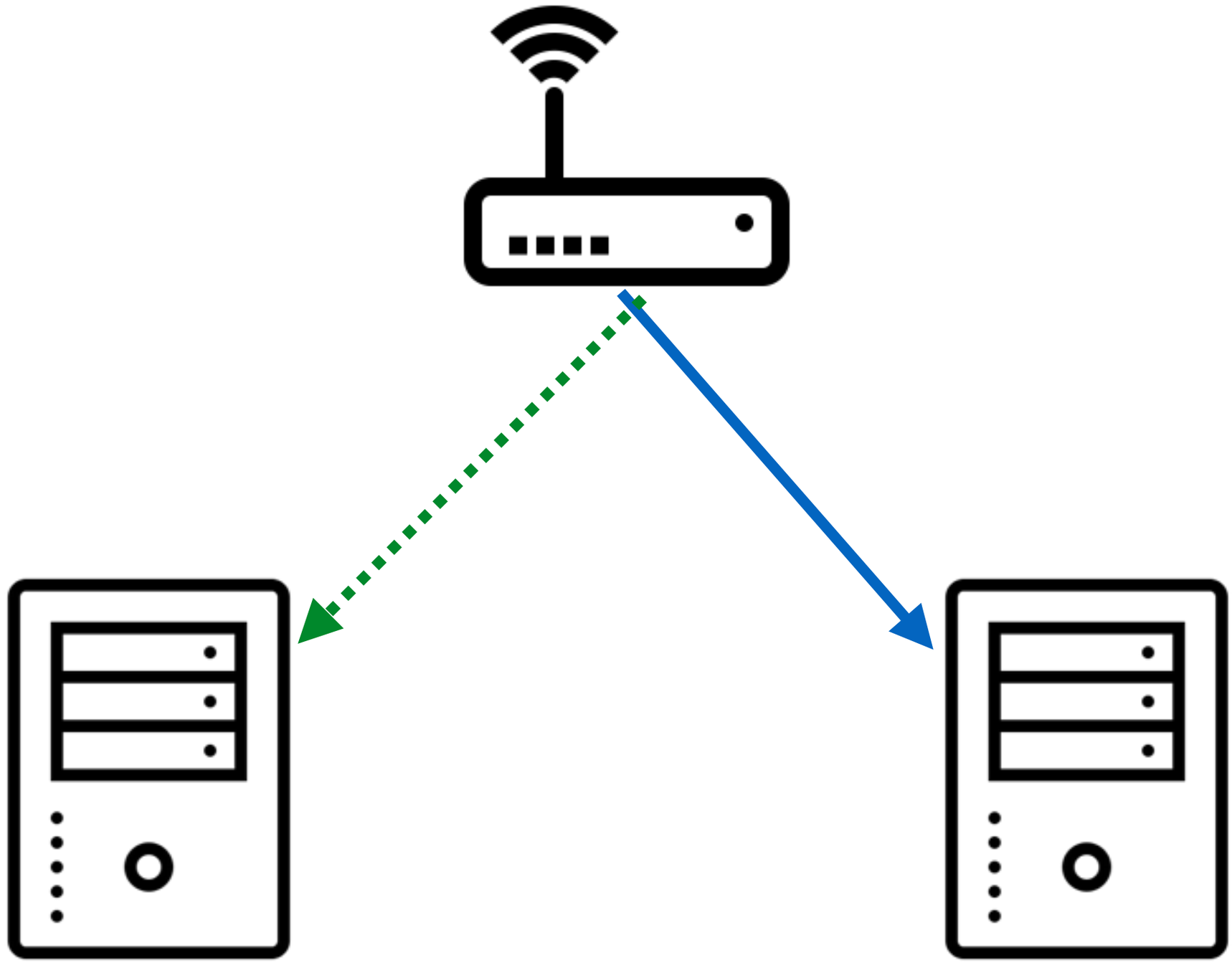
# Blue-green



# Blue-green



# Blue-green



# Lab 9:

# Make deployment blue/green

- Use ‘switch.sh’ to move traffic to 10002 (green) after deploy to 10002
- Use ‘switch.sh’ to move traffic to 10001 (blue) after deploy to 10001

# Lab 10:
# Create Job
# to Poll SCM

# Sam adds code coverage to the build

We are getting reports from users about a bug in our system.

We are getting reports from users about a bug in our system.

We are able to reproduce it.

# What is the root cause?

We are able to reproduce it.

# What is the root cause?

It appears that the “if” was tested but not the “else” for the feature.

If we added code coverage to the build we could be more confident about not introducing new issues.

It appears that the “if” was tested but not the “else” for the feature.

If we added code coverage to the build we could be more confident about not introducing new issues.

We could fail the build if our current coverage percentage drops.

# Lab 11: Add Code Coverage Metrics

```
stages {




}
```

```
stages {
  stage("cobertura") {
     steps {
      sh "mvn clean cobertura:cobertura"
     }
   }
}
```

}

```
stages {
  stage("cobertura") {
     steps {
      sh "mvn clean cobertura:cobertura"
     }
   }
}
post {
}
```

```
stages {
  stage("cobertura") {
     steps {
      sh "mvn clean cobertura:cobertura"
     }
   }
}
post {
  always {

    }
}
```

```
stages {
  stage("cobertura") {
     steps {
      sh "mvn clean cobertura:cobertura"
     }
   }
}
post {
  always {
     cobertura
  }
}
```

```
stages {
  stage("cobertura") {
     steps {
      sh "mvn clean cobertura:cobertura"
     }
   }
}
post {
  always {
     cobertura autoUpdateHealth: false,
     }
}
```

```
stages {
  stage("cobertura") {
     steps {
      sh "mvn clean cobertura:cobertura"
     }
   }
}
post {
  always {
     cobertura autoUpdateHealth: false,
     autoUpdateStability: false,
   }
}
```

```
stages {
  stage("cobertura") {
     steps {
      sh "mvn clean cobertura:cobertura"
     }
   }
}
post {
  always {
     cobertura autoUpdateHealth: false,
  autoUpdateStability: false,
  coberturaReportFile: '**/coverage.xml',
  }
}
```

```
stages {
  stage("cobertura") {
     steps {
      sh "mvn clean cobertura:cobertura"
     }
   }
}
post {
  always {
     cobertura autoUpdateHealth: false,
  autoUpdateStability: false,
  coberturaReportFile: '**/coverage.xml',
  conditionalCoverageTargets: '70, 0, 0',
  }
}
```

```
stages {
  stage("cobertura") {
     steps {
      sh "mvn clean cobertura:cobertura"
     }
   }
}
post {
  always {
     cobertura autoUpdateHealth: false,
  autoUpdateStability: false,
  coberturaReportFile: '**/coverage.xml',
  conditionalCoverageTargets: '70, 0, 0',
  failUnhealthy: false,




  }
}
```

```
stages {
  stage("cobertura") {
     steps {
      sh "mvn clean cobertura:cobertura"
     }
   }
}
post {
  always {
     cobertura autoUpdateHealth: false,
  autoUpdateStability: false,
  coberturaReportFile: '**/coverage.xml',
  conditionalCoverageTargets: '70, 0, 0',
  failUnhealthy: false, failUnstable: false,




  }
}
```

```
stages {
  stage("cobertura") {
     steps {
      sh "mvn clean cobertura:cobertura"
     }
   }
}
post {
  always {
     cobertura autoUpdateHealth: false,
  autoUpdateStability: false,
  coberturaReportFile: '**/coverage.xml',
  conditionalCoverageTargets: '70, 0, 0',
  failUnhealthy: false, failUnstable: false,
  lineCoverageTargets: '80, 0, 0',



  }
}
```

```
stages {
  stage("cobertura") {
     steps {
      sh "mvn clean cobertura:cobertura"
     }
   }
}
post {
  always {
     cobertura autoUpdateHealth: false,
  autoUpdateStability: false,
  coberturaReportFile: '**/coverage.xml',
  conditionalCoverageTargets: '70, 0, 0',
  failUnhealthy: false, failUnstable: false,
  lineCoverageTargets: '80, 0, 0',
  maxNumberOfBuilds: 0,


  }
}
```

```
stages {
  stage("cobertura") {
     steps {
      sh "mvn clean cobertura:cobertura"
     }
   }
}
post {
  always {
     cobertura autoUpdateHealth: false,
  autoUpdateStability: false,
  coberturaReportFile: '**/coverage.xml',
  conditionalCoverageTargets: '70, 0, 0',
  failUnhealthy: false, failUnstable: false,
  lineCoverageTargets: '80, 0, 0',
  maxNumberOfBuilds: 0, methodCoverageTargets:
  '80, 0, 0',

  }
}
```

```
stages {
  stage("cobertura") {
     steps {
      sh "mvn clean cobertura:cobertura"
     }
   }
}
post {
  always {
     cobertura autoUpdateHealth: false,
  autoUpdateStability: false,
  coberturaReportFile: '**/coverage.xml',
  conditionalCoverageTargets: '70, 0, 0',
  failUnhealthy: false, failUnstable: false,
  lineCoverageTargets: '80, 0, 0',
  maxNumberOfBuilds: 0, methodCoverageTargets:
  '80, 0, 0', onlyStable: false,

  }
}
```

```
stages {
  stage("cobertura") {
     steps {
      sh "mvn clean cobertura:cobertura"
     }
   }
}
post {
  always {
     cobertura autoUpdateHealth: false,
  autoUpdateStability: false,
  coberturaReportFile: '**/coverage.xml',
  conditionalCoverageTargets: '70, 0, 0',
  failUnhealthy: false, failUnstable: false,
  lineCoverageTargets: '80, 0, 0',
  maxNumberOfBuilds: 0, methodCoverageTargets:
  '80, 0, 0', onlyStable: false, sourceEncoding:
  'ASCII',
  }
}
```

```
stages {
  stage("cobertura") {
     steps {
      sh "mvn clean cobertura:cobertura"
     }
   }
}
post {
  always {
     cobertura autoUpdateHealth: false,
  autoUpdateStability: false,
  coberturaReportFile: '**/coverage.xml',
  conditionalCoverageTargets: '70, 0, 0',
  failUnhealthy: false, failUnstable: false,
  lineCoverageTargets: '80, 0, 0',
  maxNumberOfBuilds: 0, methodCoverageTargets:
  '80, 0, 0', onlyStable: false, sourceEncoding:
  'ASCII', zoomCoverageChart: false
  }
}
```

# Sam adds a dashboard

Where are my features?

Where are my features?

If the code has been checked in, it should just be a bit until it shows up in production.

Alright, I'll wait.

If the code has been checked in, it should just be a bit until it shows up in production.

Still not there!

Still not there!

That’s odd, let me check with my team.

Did you push the changes that business was asking for to master?

Did you push the changes that business was asking for to master?

Yeah, like 2 hours ago.

Hmm, did you check the build?

Yeah, like 2 hours ago.

Hmm, did you check the build?

Looks like it failed on the integration tests. Builds after it are failing too.

We just found out now?

Looks like it failed on the integration tests. Builds after it are failing too.

We just found out now?

We could add a notification system to our build or have a big visual in the team space.

We should also not push on broken builds.

We could add a notification system to our build or have a big visual in the team space.

# Dashboard example

# End of the (pipe)line

# Benefits

- Get to market faster
- Greater confidence, less stress
- Smaller changes imply less risk
- Simpler branch strategy
- Visibility

# Watch out for…

- out of shape test suites
- alienating allies
- emotional reactions to change

# For full slides and pipeline scripts:

https://github.com/jbalmert/always_be_delivering

Jimmy Balmert
twitter: @trembyl

Robert Cochran
twitter: @cochrarj